COVERAGE FOR MOBILE ROBOTS WITH UNCERTAINTY

BY

COLIN E. DAS

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Aerospace Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Adviser:

Professor Timothy Bretl

Abstract

The classical problem of robot coverage is to plan a path that brings a point on a robot within a fixed distance of every point in the free space. This problem is solved under the assumption that a robot follows the path exactly. However, a robot with uncertainty in sensing and actuation is not guaranteed to follow the path exactly and thus is not guaranteed to cover the free space. A coverage planner, rather than producing a path, should produce a feedback policy, but it is unclear exactly what performance it is to achieve. To make the problem concrete, we formulate a "probably approximately correct" measure of performance that captures the probability $1 - \varepsilon$ of covering a fraction $1 - \delta$ of the free space.

We apply this measure to a simple system in which a robot is commanded to follow a square wave path in a rectangular region. We show that following a square wave path with passes parallel to the short axis performs better than following a square wave path with passes parallel to the long axis. This inspires a new strategy in which a square wave path is steered throughout the free space. We design a feedback policy where the square wave path is treated as a larger, *virtual* robot with coverage implement size equal to the size of the square wave. The virtual robot tracks a global path and generates a local path that the robot tracks. Coverage planning is then simplified by decoupling local and global coverage strategies. We describe in detail the coverage planning algorithms for the robot and the virtual robot, specifically the calibration, estimation, and control processes and implement the algorithm in simulation.

For the particular case of the virtual robot operating along the boundary, we introduce a method of simultaneous coverage and calibration. For robots with boundary sensors that trigger when entering or leaving the free space, like the ones we consider, we show that it is possible to estimate the robot odometric parameters based on the difference in time between boundary crossings. Through an analysis of observability, we design a sequence of switchback trajectories that produce robust parameter estimations. This algorithm is implemented both in simulation and hardware experiment.

To my parents, for their love and support.

Acknowledgments

This would not have been possible without the members of my research group, including my adviser Professor Timothy Bretl. Aaron Becker and Miles Johnson especially aided me throughout this project and mentored me throughout my graduate studies. Abdullah Akce helped greatly with simulations and Dennis Matthews with hardware modifications. Navid Aghasadeghi, Aadeel Akhtar, and Erik Kroeker, and Cem Onyuksel consistently provided a great platform for testing ideas.

I would also like to thank Daniel Block for his experimental advice and willingness to help repair the robots and Professor Dušan Stipanović and Chad Burns for their lab space. Finally, I would like to thank Catherine Mao and Hero for their kindness and support.

The professors and teachers I have encountered at the University of Illinois have taught me the excitement and challenge of engineering.

Table of Contents

List of Abbreviations

EKF	extended Kalman filter
LQG	linear quadratic gaussian
LQR	linear quadratic regulator
OC	online calibration
PI	proportional integral
SL	straight line
SLAC	simultaneous localization and calibration
VR	virtual robot
ZPT	zero-point turn

Chapter 1

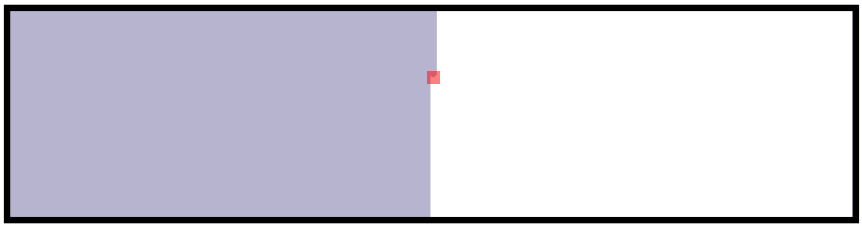
Introduction

The problem of robot coverage is to plan a path that brings a point on a robot within some fixed distance of every point in the free space. The "fixed distance" in this problem is the size of the robot coverage implement, for example the radius of a mower blade. The "free space" is the region of the workspace that is to be covered, for example a yard excluding trees and sidewalks. This problem has been the focus of considerable research over the past several decades (e.g., see [1]–[33]) and is important for a variety of military, industrial, and domestic applications that include painting, demining, floor cleaning, and lawn mowing.

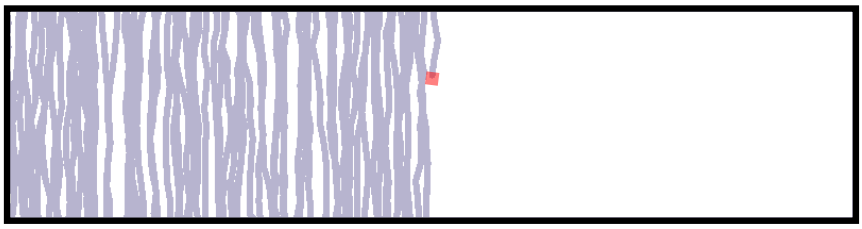
Modern coverage planners can guarantee that a path swept out by a robot completely covers the free space, but only if the robot follows the path with negligible error. In many cases, this assumption is valid. For example, both the end-effector of an industrial robot arm and an autonomous tractor with centimeter-level GPS can achieve negligible error in global position. However, in the presence of significant uncertainty in sensing and actuation, in may no longer be possible to guarantee complete coverage in finite time [34]. For example, a robot with consumer-grade GPS in an outdoor environment often has localization error variance larger than the coverage implement leading to worse performance. Fig. 1.1 shows results from a simulated robot with and without uncertainty.

The optimal, or minimum-time, solution is to move in such a way that the robot always covers new territory until completion. While the globally optimal solution is unknown, sub-optimal solutions exist [35]. Much focus has been on determining provably complete paths, typically generated using cellular decomposition. The coverage task becomes difficult as sensors degrade and as the motion model uncertainty grows. Variations in the terrain such as slope and friction coefficients, aging and wear of the robot, improper calibration, robot-terrain interactions such as wheel load-up and wheel sinkage, and inaccurate localization sensors all are potential sources of uncertainty.

The conventional approach to deal with this uncertainty is to follow a random path while staying in the boundaries of the free space. This is an exhaustive approach: the longer the robot runs, the more likely it is that the area is covered. This approach has two significant downfalls: (1) it is inefficient to double back over already-covered ground and (2) the robot



(a) coverage in a system with negligible uncertainty in sensing and actuation



(b) coverage in a system with significant uncertainty in sensing and actuation

Figure 1.1: Equal time simulations of a robot attempting to cover a rectangular area by following a square wave path in a system with negligible uncertainty (top) and a system with significant uncertainty (bottom). A nominal path for full coverage without uncertainty experiences a degradation of coverage under measurement and process noise. The goal of this work is to analyze and improve the performance of coverage policies for such systems.

can be caught in parts of the yard.

A robot with uncertainty in sensing and actuation is not guaranteed to follow a path exactly and thus is not guaranteed to cover the free space. A coverage planner, rather than producing a path, should produce a feedback policy, but it is unclear exactly what performance it is to achieve because it may no longer be possible to guarantee that the robot covers all of the free space in finite time [34]. Prior art has provided several measures of performance such as the expected fraction covered, the variance of fraction covered, and the probability that the entire free space is covered. One goal of this work is to unify these candidate metrics.

To make the problem concrete, we formulate a "probably approximately correct" measure of performance that captures the probability $1 - \varepsilon$ of covering a fraction $1 - \delta$ of the free space:

$$P(C \geq 1 - \delta) \geq 1 - \varepsilon \qquad \varepsilon, \delta \in [0, 1].$$

The problem of coverage for a robot with uncertainty is then to plan a feedback policy that achieves a given value of ε and δ. Just as solutions to the classical problem are judged by the resulting path length, solutions to our problem are judged by the required execution time. We apply this performance measure to an example system and illustrate its usefulness

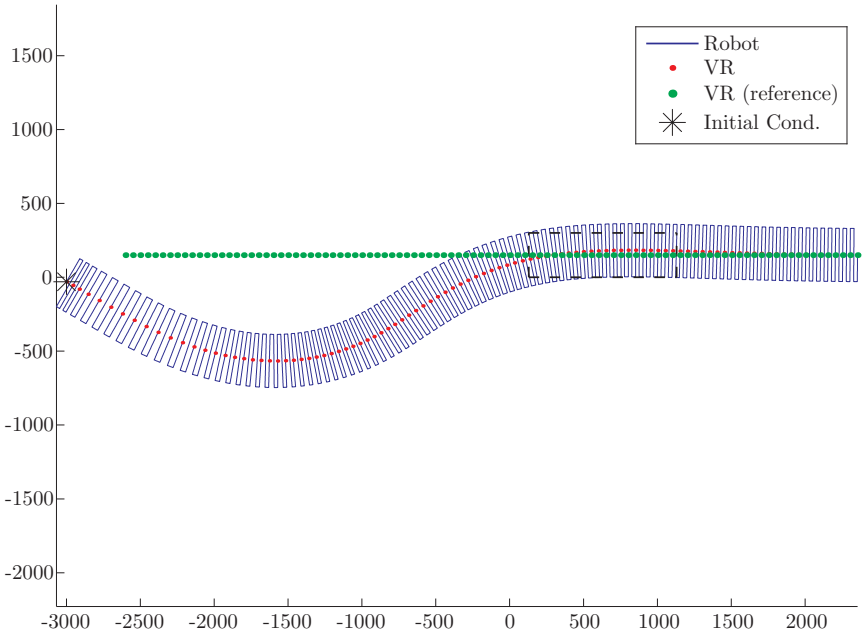


Figure 1.2: Simulation of the virtual robot tracking a straight line trajectory from arbitrary initial conditions. Units are arbitrary. The rectangular region (black dashed line) is a desired region to be covered. No process or measurement noise is injected.

by ranking policies with respect to the probability of achieving a desired fraction covered, computing the minimum time required to meet a given performance objective, and guiding robot sensor selection.

One result from this analysis is for a robot under uncertainty commanded to cover a rectangular region with a square wave path. We show that following a square wave pattern with passes parallel to the short axis performs better than following a square wave pattern with passes parallel to the long axis. This inspires a new strategy in which a square wave is steered throughout the free space. We design a feedback policy where the square wave path is treated as a larger, *virtual* robot with coverage implement size equal to the size of the square wave. In fact, we model the virtual robot with the same equations of motion, but with kinematics dependent on the shape of the square wave. An example is given in Figure 1.2.

The virtual robot simplifies coverage planning by isolating local and global strategies. By using a single instance of a square wave as a motion primitive, we model the inputs from the virtual robot trajectory tracker as path parameters that shape the primitive and thus the robot trajectory. Though state estimates are inherently coupled through one set of measurements, separate controllers are used for each robot. The virtual robot follows a global path and, in doing so, generates a local path that the robot follows.

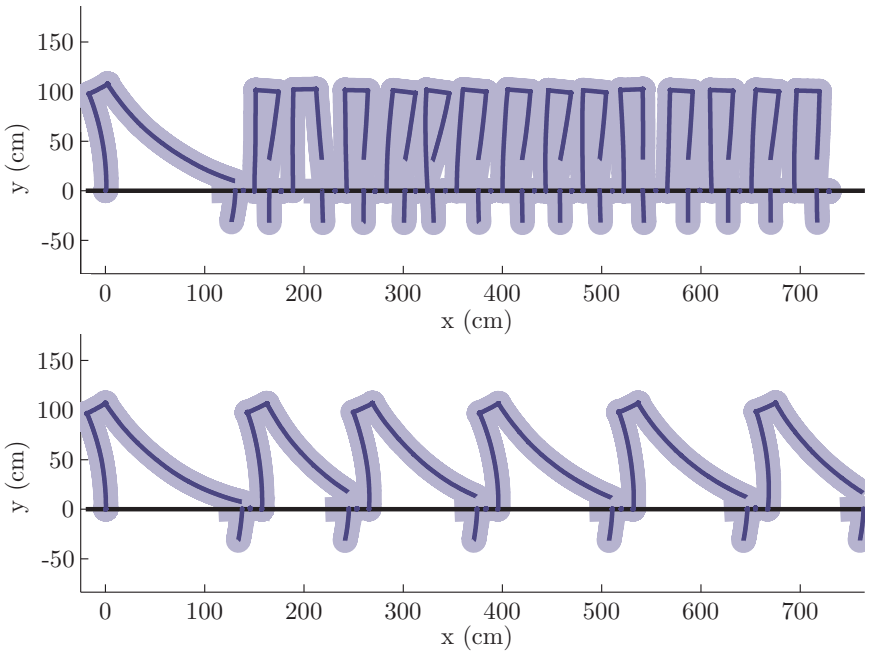


Figure 1.3: The result in simulation of applying our approach to simultaneous calibration and coverage. The robot is attempting to execute a rectangular coverage pattern consisting of multiple switchbacks. Coverage is shown in light blue and the movement primitives are shown in dark blue. The boundary line is in black at $y = 0$. The robot initially uses nominal *miscalibrated* parameters. A robot using online calibration learns effective parameters during coverage, leaving little uncovered white space (top). A robot that does not use online calibration exhibits poor coverage (bottom).

In implementing such a policy, we require accurate models of both the robot and the virtual robot. The robot is a standard differential drive robot and the virtual robot has similar equations of motion, but with kinematics that are coupled with the square wave trajectory. These models must be calibrated for accurate location estimation and control. That is, we must identify the robot odometric parameters—the wheel sizes and the wheel separation—and compute bounds on the virtual robot path parameters so that the local policy achieves the desired performance objective. We present several well-known methods of offline calibration such as the UMBmark test [36] and the extended Kalman filter (EKF). A linear quadratic Gaussian (LQG) optimal controller computes the state estimate and control input for the robot. The virtual robot computes its state estimate from the robot state and uses a linear quadratic regulator (LQR) controller to track the global path.

As an extension, we consider the particular case of the virtual robot operating along the boundary. We introduce a method of simultaneous calibration and coverage near the boundary for a robot with access to a boundary sensor that triggers when entering or leaving

the free space. We show that it is possible to estimate the robot odometric parameters based on the difference in time between boundary crossings. Through an analysis of observability, we design a sequence of switchback trajectories that produce robust parameter estimations. This algorithm is implemented both in simulation and hardware experiment. Figure 1.3 shows the effect of the online calibration algorithm.

This thesis is organized as follows. We begin with a discussion of the problem of coverage under uncertainty (Chapter 2). This is followed by a description of the virtual robot concept and algorithm (Chapter 3). Then, we present the process and sensing models of the robot and the virtual robot (Chapter 4). We explain the methods of calibration for the robot (Chapter 5). Then, we provide the estimation and control processes used in the algorithm (Chapter 6). We implement the virtual robot algorithm in simulation (Chapter 7). Then, we extend the virtual robot policy to online calibration near the boundary (Chapter 8). Finally, we discuss broader implications in our concluding remarks (Chapter 9).

Chapter 2

Coverage

Conventional motion planning focuses on the problem of *navigation*—determining a collision-free path from a start configuration to a goal configuration. The classical problem of robot coverage is to plan a path that—if followed exactly—would bring a point on the robot within a fixed distance of every point in the free space. See Figure 2.1. The coverage problem has been the focus of considerable research over the past three decades and is important for a variety of applications that include painting, demining, floor cleaning, and lawn mowing.

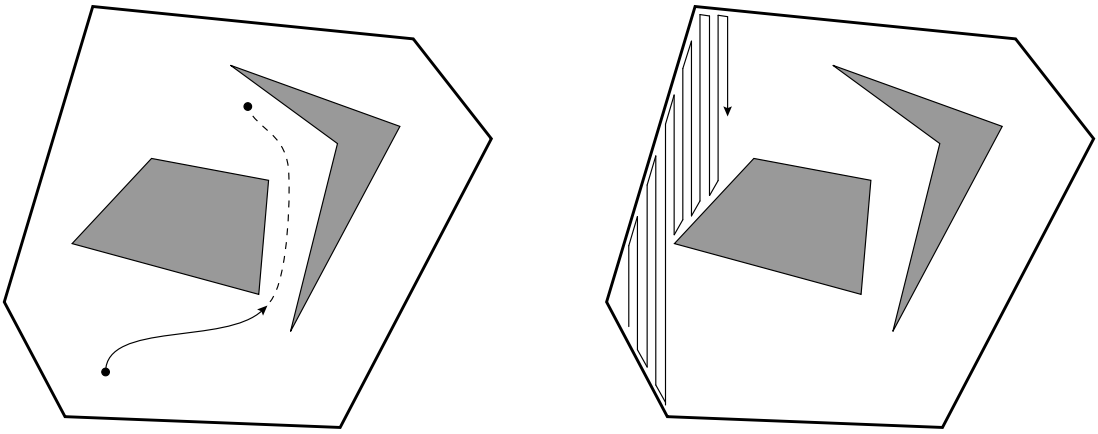
In many cases, we can assume that the path planned by a coverage algorithm is followed by the robot, if not exactly, then at least with negligible error. Such an assumption would be reasonable for an industrial robot arm or an autonomous tractor with centimeter-level GPS. Provably complete solutions to the coverage problem exist under the assumption that a robot follows the path exactly.

However, a robot with uncertainty in sensing and actuation is not guaranteed to follow the path exactly and thus is not guaranteed to cover the free space. For example, a robot with consumer-grade GPS in an outdoor environment often has localization error variance that is bigger than the size of its coverage implement, causing parts of the free space to be missed and leading to performance of the sort shown in Fig. 1.1.

A coverage planner, rather than producing a path, should produce a feedback policy, but it is unclear exactly what performance it is to achieve because it may no longer be possible to guarantee that the robot covers all of the free space in finite time [34]. Prior art has provided a variety of ways to measure performance, including by the expected fraction covered, the variance of fraction covered, and the probability that the entire free space is covered. Our goal in this chapter is to introduce a more general measure of performance that unifies these candidate metrics.

To make the problem concrete, we formulate a "probably approximately correct" measure of performance that captures the probability $1 - \varepsilon$ of covering a fraction $1 - \delta$ of the free space.

$$P(C \geq 1 - \delta) \geq 1 - \varepsilon \qquad \varepsilon, \delta \in [0, 1].$$



(a) navigation (b) navigation

Figure 2.1: Conventional motion planning focuses on navigation—determining a collision-free path from a start configuration to a goal configuration (left). Applications such as floor cleaning, demining, and lawnmowing require covering the entire free space (right).

The problem of coverage for a robot with uncertainty is then to plan a feedback policy that achieves a given value of ε and δ. Just as solutions to the classical problem are judged by the resulting path length, solutions to our problem are judged by the required execution time. We apply this performance measure to an example system and illustrate its usefulness by ranking policies with respect to the probability of achieving a desired fraction covered, computing the minimum time required to meet a given performance objective, and guiding robot sensor selection.

The remainder of this chapter proceeds as follows. We begin by presenting a brief review of related work, focusing on existing measures of performance for coverage under uncertainty (Section 2.1). Then, we present our alternative PAC measure of performance (Section 2.2.1), and consider its application to several examples in simulation (Section 2.2.4).

2.1 Related Work

We give a brief overview of robot coverage and highlight research on coverage under forms of uncertainty. We also survey coverage statistics relevant to robotics.

2.1.1 Robot Coverage

Robot coverage planning algorithms typically use a form of *cellular decomposition* which breaks up the free space into simple regions, called *cells*, whose union fills (or approximately fills) the free space. Planners with an exact cellular decomposition achieve a provable guarantee of complete coverage if the robot visits each cell. Exact cellular decomposition can be achieved using on-board sensing via the *boustrophedon* (a square wave path) decomposition [4] or Morse functions [6], both of which rely on the identification of *critical points* [7], points where the connectivity of a slice moving across the workspace changes. In these works the goal is to construct a minimum-length path that visits and covers each cell. Choset showed that the minimum path length is bounded linearly by the area of the free space, the number of critical points, and the perimeter lengths of the obstacles and outer boundary [11]. In the following section, we modify the boustrophedon path so that the reference trajectory has path spacing less than the width of the coverage implement and extends beyond the boundary. These are heuristic methods of dealing with uncertainty, but are simply used as candidate policies.

While [6] and [7] achieve complete coverage under sensing uncertainty, their focus is on rejecting bad sensor readings. They assume that most readings are good and that bad readings are exceptional. In the following section, we consider systems where all sensor readings and actuations are corrupted by Gaussian noise. Algorithms such as [37] and [38] deal with robotic uncertainty, but consider the navigation problem—moving from one configuration to another—not the coverage problem.

Algorithms in [37] and [38] deal with robotic uncertainty by determining regions from which particular motions are guaranteed to reach a desired goal. Regions are constructed by modeling position uncertainties as geometric constraints. Constraints on regions from which particular motions are guaranteed to reach a goal correspond to bounds on uncertainties. These represent the navigation problem of moving to a desired final configuration. We focus on motion plans to achieve (or partially achieve) coverage, which is the aggregate result of following the entire path.

We are specifically interested in systems that do not have a *completion detector*—that is, systems that cannot sense when the coverage task has been completed. Systems with a completion detector, such as an overhead monitoring system [12], "ant"-type robots that leave a detectable trail over covered cells [13], or perfect localization [5] form a different class of problem.

For random-reflection, [9] analyzes policy performance in detecting 80% of mines found versus the total mean search time. The data represents performance over a discrete number

of points, i.e., mines, in the free space and is expressed by mean and standard deviation values. If the mines are uniformly distributed, this problem is a special case of the coverage problem we consider.

2.1.2 Coverage Statistics

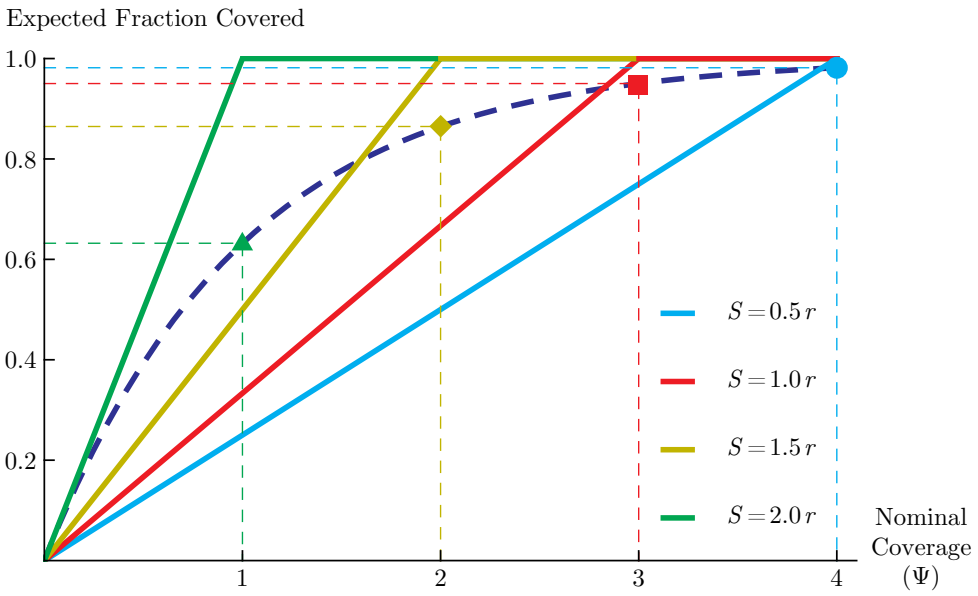
Prior art has investigated coverage for many end purposes, resulting in a variety of performance objectives for coverage policies. The ideal performance measure is the probability density function (PDF) of coverage for a given policy because any other statistic can be constructed from the PDF. This ideal has so far proved illusive. WWII led to a surge in research funding for coverage related to search and rescue, reconnoissance, bombing, weapon salvos and mine-sweeping. This research generated the expected coverage probabilities for both random and boustrophedon search paths. Later work led to additional statistics about the probability distributions for coverage tasks.

As shown in [39], the expected fraction covered $\mathbb{E}[C(t)]$ of a region with area A by a random strategy in time t is

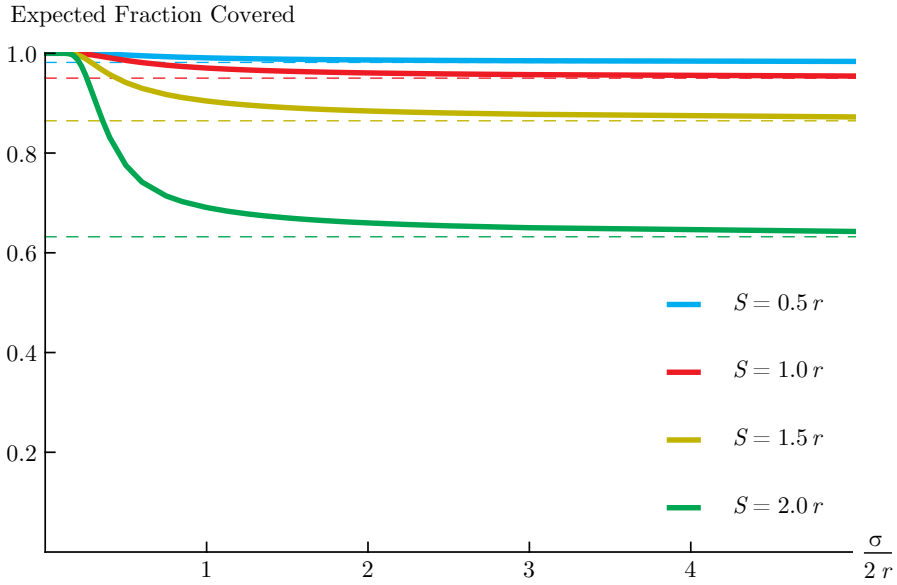
$$\mathbb{E}[C(t)] = 1 - e^{-\frac{2rvt}{A}}. \tag{2.1}$$

for velocity v, and a coverage implement radius r. They also considered search via boustrophedon paths under three simplifying assumptions: each pass is long enough that the paths may be considered independent, the paths are corrupted by a zero-mean Gaussian noise with standard deviation σ, and the number of passes is large compared to the width of the area to be covered. With these assumptions, the function $\mathbb{E}[C(r, S, \sigma)]$ is the expected fraction covered after following a boustrophedon path with path centers spaced S units apart. It is computed by first calculating $g(i, x)$, the probability of a point at x being covered by the i-th pass; next finding $h(x)$, the total probability of coverage at x after parallel passes $i = -\infty$ through $i = \infty$; and finally computing the average coverage by integrating $h(x)$ over one path width.

$$\begin{aligned} g(i,x) &= \frac{1}{\sqrt{2\pi}} \int_{(x-iS-r)/\sigma}^{(x-iS+r)/\sigma} e^{-\frac{1}{2}z^2}\, dz \\ h(x) &= 1 - \prod_{i=-\infty}^{\infty} \big(1 - g(i,x)\big) \\ \mathbb{E}[C(r,S,\sigma)] &= \frac{1}{S} \int_{-S/2}^{S/2} h(x) \end{aligned} \tag{2.2}$$



(a) Expected fraction covered by a random strategy (dashed blue) and four boustrophedon paths with different path spacing (S) and zero uncertainty ($\sigma = 0$). The markers indicate the expected fraction covered by a random strategy after executing for equal times as the boustrophedon paths.



(b) Expected fraction covered versus the ratio of the uncertainty (σ) to the coverage implement size ($2r$) for boustrophedon paths with different path spacing (S). As the uncertainty increases (x-axis), the expected fraction covered converges to that of a random strategy executing for equal time (the markers in Fig. 2.2(a)).

Fig. 2.2(a) shows the expected fraction covered versus time for a random path and for a boustrophedon path with several values of path spacing and no uncertainty ($\sigma = 0$). Time is expressed in units of *nominal coverage* Ψ, the time required to fully cover an equal area with no overlap. Fig. 2.2(b) shows the expected fraction covered versus uncertainty σ for the same path spacings as in Fig. 2.2(a). As σ increases, the $\mathbb{E}[C(\cdot)]$ converges to that of a random strategy traveling an equal distance. With (2.1) or (2.2) we can use the Markov inequality to find a loose upper bound of the form $P(C \geq X) \leq \mathbb{E}[C]/X$. To obtain tighter bounds, we need additional information.

The probabilities for vacancy and coverage of a space in $\mathbb{R}^n$ by random n-D spheres is a topic of considerable interest to the mathematics community [34, 40], but the full probability distribution function is only known for the 1-D case. We focus on 2-D results because of their relation to robotic coverage. The mean and variance for coverage of a plane under the *torus convention*[1] by discs whose centers are distributed randomly are

$$\begin{aligned} \mathbb{E}[C] &= 1 - e^{-\frac{n\pi r^2}{A}} \\ \sigma^2(C) &= A\pi e^{-\frac{2n\pi}{A}} \left(8 \int_0^1 \left(e^{\frac{nB(x)}{A}} - 1 \right) x \, dx - \frac{n\pi}{A} \right) \end{aligned} \tag{2.3}$$

where

$$B(x) = \begin{cases} 0 & \text{if } x > 1 \\ -x\sqrt{1-x^2} + \operatorname{acos}(x) & \text{otherwise} \end{cases}$$

As the ratio of disc size to region area decreases, the contribution by the torus convention asymptotes to zero.

Simulation shows that (2.3) well describes the performance of a random policy. Indeed, the expected values given by (2.1) and (2.3) are identical. Having the variance allows us to obtain tighter bounds on coverage estimates using Chebyshev's inequality. Unfortunately, such results do not exist for the boustrophedon policy.

The probability of completely covering a region by randomly placing *shapes* is given in [41]. Shapes are placed isotropically with centers inside the region. Complete coverage has a Bernoulli distribution with success probability p, which increases as the number of shapes n increases, the region area A decreases, or the areas a of the shapes increase. The expected number of gaps (uncovered areas) less the number of isles (unattached covered areas) can

[1]The *torus convention* treats a rectangular area as a torus such that any disc that protrudes out one side of the rectangle enters again from the opposite side.

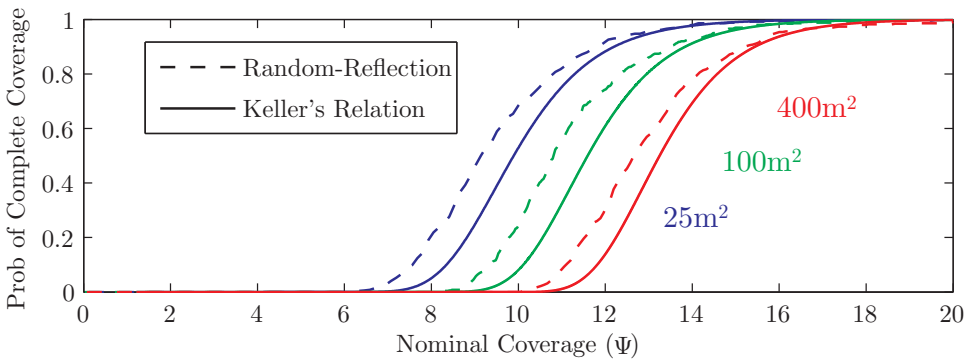


Figure 2.2: Complete coverage results for simulated random-reflection and (2.4). Solid lines represent 500 simulations of the random-reflection policy for three region sizes with a coverage implement radius r. Dashed lines show coverage on the same regions by placing $2r \times 1\text{m}^2$ rectangles randomly with centers inside the region. Note that due to continuity constraints random-reflection outperforms a purely random strategy.

be computed exactly:

$$\Phi = e^{-\Psi}\left(\Psi\left(\frac{sSa}{2\pi} - \frac{\chi A}{a}\right) + \frac{\Psi^2 A s^2}{4\pi a^2} + X\right) \tag{2.4}$$

Here s and S are the perimeter lengths of a and A, $\Psi = na$ is the nominal coverage if the n shapes did not overlap, and χ and X are the Euler characteristic of the shapes a and A, which for simple shapes without holes is 1.

For large n, the probability for isles becomes vanishingly small, and Φ represents the expected number of gaps. Complete coverage occurs when there are zero gaps. A valid approximation of complete coverage for large n is $1-e^{-\Phi}$. This function compares reasonably with a random-reflection policy, as shown in Fig. 2.2. Due to continuity constraints, random-reflection outperforms a purely random strategy for the same nominal coverage. Sadly, no similar results exist for a boustrophedon policy.

2.2 Significant Uncertainty

2.2.1 Performance Measure

A robot under significant uncertainty in sensing and actuation may not be able to guarantee that it covers all of the free space all the time, and so it becomes unclear what problem we are trying to solve. The breadth of the coverage problem with uncertainty is illustrated by two coverage tasks: painting and demining. Painting represents tasks that are considered a

success only if the region is completely covered. Demining represents tasks where success is achieved when a robot guarantees that some fraction of the free space has been covered. A ready example of such a criterion is given by the UN mine-clearing standard of 1996, requiring 99.6% of an area be checked for mines in order to be "cleared" [42]. Our performance measure has a more general form so it can model both painting and demining.

Previous work on coverage provides several performance measures such as the expected fraction covered for a given distance traveled (2.1)–(2.3) and the probability of complete coverage (2.4). These measures provide only a single point of comparison. A better performance measure would allow us to compare the entire probability distribution.

We define such a measure by:

$$P(C \geq 1 - \delta) \geq 1 - \varepsilon \tag{2.5}$$

where C is the fraction covered and $\varepsilon, \delta \in [0, 1]$. This is a "probably approximately correct" measure of performance that captures the probability $1 - \varepsilon$ of covering a fraction $1 - \delta$ of the free space [43],[44]. Certain values of ε, δ correspond naturally with common robotic coverage tasks. A painting robot, required to cover an entire space with probability $1 - \varepsilon$, has $\delta = 0$. In contrast, a demining robot that achieves success when it is guaranteed to cover a fraction $1 - \delta$, has $\varepsilon = 0$.

For any policy, the cumulative distribution function (CDF) captures the full range of performance. For a given system and policy, the CDF can either be constructed directly from the generating probability distribution or approximated by Monte Carlo analysis. Because the generating probability distributions are unknown for 2-D, as shown in Section 2.1.2, we construct the CDF from Monte Carlo simulations.

The relationship between the PAC objective and the CDF is shown by the following transformation:

$$\begin{aligned} P\left(C \geq 1 - \delta\right) &\geq 1 - \varepsilon \\ \int_{1-\delta}^{1} \mathrm{PDF}(\tau)\, d\tau &\geq 1 - \varepsilon \\ 1 - \int_{0}^{1-\delta} \mathrm{PDF}(\tau)\, d\tau &\geq 1 - \varepsilon \\ 1 - \mathrm{CDF}(1 - \delta) &\geq 1 - \varepsilon \end{aligned}$$

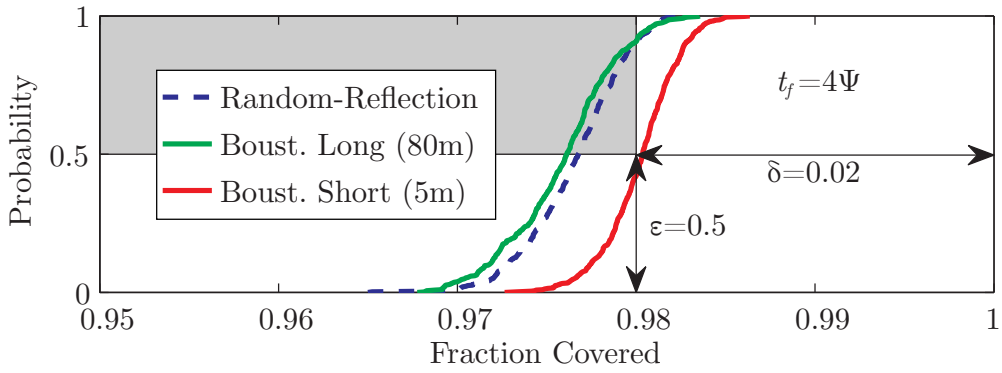


Figure 2.3: Cumulative distribution functions for three policies with an execution time $t_f = 4\Psi$ in a rectangular region ($W = 80$m, $H = 5$m). The grey rectangle depicts the desired performance constraint of ($\varepsilon = 0.5, \delta = 0.02$)—the robot must "cover at least 98% of the free space with probability at least 50%". The boustrophedon short policy exceeds the requirement, but the random-reflection and boustrophedon long policies fail. All CDFs represent 500 simulations, each with process noise $\sigma_p = 0.1$m and measurement noise $\sigma_m = 1.0$m.

Equivalently,

$$\text{CDF}(1-\delta) \leq \varepsilon. \tag{2.6}$$

This is the constraint that for a given δ, CDF$(1-\delta)$ must be less than or equal to ε. A desired performance, characterized by an ε,δ pair, places a constraint rectangle on the CDF from $(0, 1)$ to $(1-\delta, \varepsilon)$. To satisfy such a constraint, the CDF must not cross the boundary of this rectangle. Fig. 2.3 shows an example of this type of constraint.

Multiple ε,δ constraints may be concatenated to fully specify the desired coverage task. For instance, if we must never have less than 50% covered and also require at least 70% covered in 8 out of 10 trials, our constraint set is $\{(\varepsilon_1 = 0, \delta_1 = 0.5) \cup (\varepsilon_2 = 0.2, \delta_2 = 0.3)\}$. This set is illustrated in Fig. 2.7. The most stringent constraint set ($\delta = 0, \varepsilon = 0$) is achieved by systems with negligible error. The random-reflection and boustrophedon policies achieve this constraint with probability 1 as time goes to infinity. In practice we are interested in tight constraints for finite-horizons.

As (2.6) shows, the CDF gives an ε-value for every δ, generating a performance curve $\varepsilon(\delta)$. It is now possible to formulate well-posed problems, e.g., finding a policy that maximizes some objective function. An example is to find a policy that maximizes the probability of achieving a desired fraction covered given a system and an execution time. That is, given a system

$$\underset{\pi \in \{\pi_1, \pi_2, \ldots, \pi_n\}}{\arg\min} \quad \varepsilon(\pi)\big|_{\delta, t_f}. \tag{2.7}$$

Another example is to find the minimum execution time that meets some desired performance. That is, given a policy, a desired ε,δ pair, and a system

$$\text{minimize } t_f \text{ such that } \text{CDF}(1-\delta) \leq \varepsilon \tag{2.8}$$

The PAC performance measure is a tool for analyzing feedback polices for coverage in systems with sensing and actuation uncertainty.

We apply the PAC measure to a simplified system under three feedback policies: random-reflection, boustrophedon long, and boustrophedon short. All three policies are employed in practice [45],[39]. We begin by describing the robotic system and the candidate policies, then describe three applications. The policies are tested on identical free spaces to allow unbiased comparison.

2.2.2 Example System

We consider discrete time, bounded control systems with white Gaussian noise in a bounded 2-D free space. Our process model is described by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{u}_k \Delta t + \boldsymbol{w}_k$$

where $\boldsymbol{x}_k, \boldsymbol{x}_{k+1} \in \mathbb{R}^2$ are system states at times $k, k+1$, $||\boldsymbol{u}_k|| \leq v$ the control input, and $\boldsymbol{w}_k \sim \mathcal{N}\left(0, \sigma_p^2\right)$ zero-mean Gaussian process noise. The measurement model at time $k+1$ is given by

$$\boldsymbol{z}_{k+1} = \boldsymbol{x}_{k+1} + \boldsymbol{\nu}_{k+1}$$

where $\boldsymbol{\nu}_{k+1} \sim \mathcal{N}\left(0, \Sigma_m^2\right)$ is the zero-mean Gaussian measurement noise and Σ_m is the diagonal covariance matrix

$$\Sigma_m = \begin{bmatrix} \sigma_m & 0 \\ 0 & \sigma_m \end{bmatrix}.$$

For each component of the state, the measurement variance is σ_m. For constant σ_p^2 and σ_m^2, the steady-state localization variance is

$$\sigma^2 = \frac{\sigma_p^2}{2}\left(-1 + \sqrt{1 + 4\left(\sigma_m/\sigma_p\right)^2}\right).$$

We are particularly interested in systems with σ^2 such that a robot attempting to follow a nominal path is not guaranteed to achieve complete coverage. The robot is also equipped with a boundary sensor to prevent it from exiting the free space.

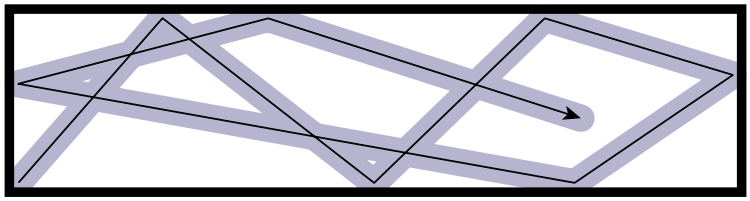


Figure 2.4: A representative path of the random-reflection policy. The robot moves until it detects a boundary, turns to a uniformly random heading, and repeats the process until time t_f.

We consider obstacle-free, rectangular workspaces with width W and height H. The free space is a bounded subset of $\mathbb{R}^2$ with no holes and is thus a single connected component. This class of free space has a simple topology and isolates problems due to uncertainty from the narrow passage problem.

2.2.3 Candidate Policies

A policy

$$\pi : \boldsymbol{x}_0, \boldsymbol{y}_0 \cdots \boldsymbol{y}_k, t_k \rightarrow \boldsymbol{u}_k$$

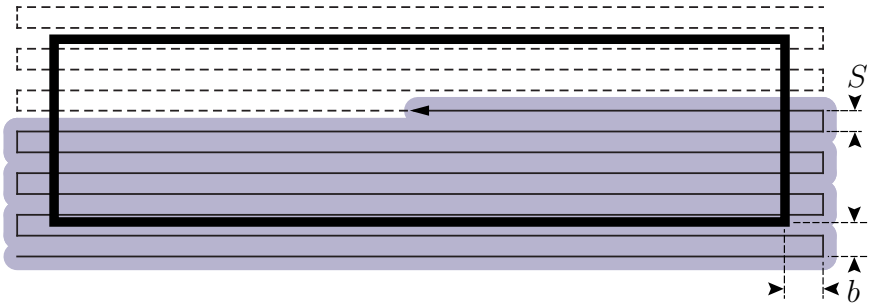
assigns the current input $\boldsymbol{u}_k$ given the initial condition $\boldsymbol{x}_0$ and the measurement history $\boldsymbol{y}_0 \cdots \boldsymbol{y}_k$.

Random Reflection

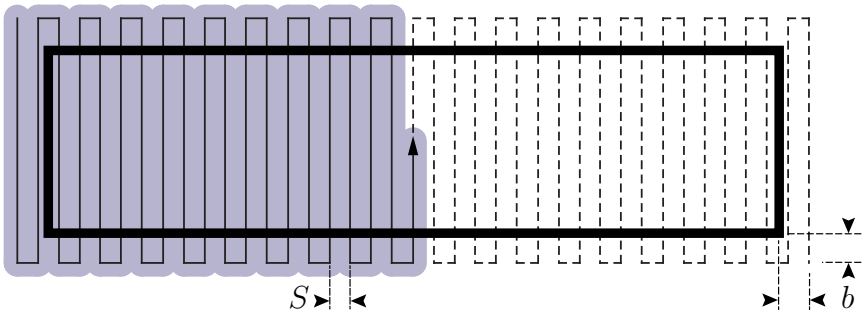
A common industry approach to coverage under uncertainty is random-reflection [9]. It requires no a priori information of the environment nor global localization sensors, but does require a boundary sensor. The robot chooses a uniformly random heading and attempts to move straight with speed v until it detects a boundary at which point the process repeats until time t_f. A representative path is shown in Fig. 2.4.

Boustrophedon (Long and Short)

The boustrophedon long and short policies are motivated by [4]. They command the robot to follow a square wave trajectory as shown in Figs. 2.5(a) and 2.5(b). The boustrophedon long policy assigns a reference trajectory of square wave motions with passes aligned with the long axis of the boundary. The boustrophedon short policy is similar, but with passes aligned with the short axis of the boundary. The trajectory is parameterized by path spacing



(a) Boustrophedon Long Policy



(b) Boustrophedon Short Policy

Figure 2.5: The boustrophedon path for policies (2) and (3). The geometry of the path is parameterized by path spacing S and overshoot b, which are the same in both x- and y-directions. The robot computes its state estimate using a KF and tracks its trajectory with an LQR controller.

S and by path overshoot b. Upon completion of the reference trajectory, the robot retraces it in reverse until time t_f. The robot uses a Linear Quadratic Regulator (LQR) controller as an optimal dynamic regulator about the reference trajectory and a Kalman Filter as a state estimator. The robot uses its boundary sensor to halt on the boundaries.

2.2.4 Examples

Ranking Policies

In this example, we use ε,δ curves to compare the three policies from Section 2.2.3. Policies are ranked by their probability of achieving a desired fraction covered of 97.5% ($\delta = 0.025$). The system is the same for all cases with process noise $\sigma_p = 0.1$m and measurement noise $\sigma_m = 1.0$m for a region with width $W = 5$m and height $H = 80$m. Applying (2.7) for $t_f = 4\Psi$, the boustrophedon short policy performs better than random-reflection and boustrophedon long policies, implying that in many cases it is better to align passes with the

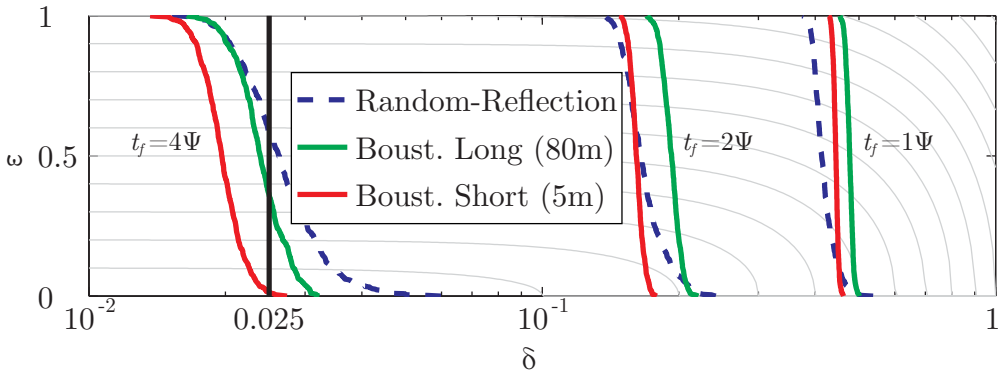


Figure 2.6: Performance curves (lower is better) for three policies in a system with uncertainty ($\sigma_p = 0.1, \sigma_m = 1.0$) in a rectangular region ($W = 5\text{m}, H = 80\text{m}$). Policies are ranked by their probability of achieving $\delta = 0.025$ with an execution time $t_f = 4\Psi$ using (2.7). The boustrophedon short policy has the highest probability ($\varepsilon = 0.01$), followed by boustrophedon long ($\varepsilon = 0.36$), and random-reflection ($\varepsilon = 0.59$).

short axis of a boundary than with the long axis. This is clear from Fig. 2.6 which shows the performance curves of each policy for $t_f = \{1, 2, 4\}\Psi$.

An implementation of this result is simply to build upon the work of [4]. That is, compute the cell decomposition and thus the adjacency graph for a given space. Then, for the given system, determine the execution time and policy which achieves some desired performance for a single cell. Then, instead of executing the back-and-forth motions as in Section 6.3.3 (which assumes the robot follows the path with negligible error), the robot should follow the appropriate policy for each cell. The values of S and b may be tuned with the policy.

Optimizing Path Length

To minimize wear-and-tear on the robot, it is often desirable to know the minimum execution-time to achieve some performance. Consider a random-reflection policy with t_f the design parameter and a desired performance of $(\varepsilon_1 = 0.0, \delta_1 = 0.5) \cup (\varepsilon_2 = 0.2, \delta_2 = 0.3)$—that the robot must "cover 50% or more of the free space every time and 70% or more with probability at least 80%." In Fig. 2.7 we show the performance for $t_f = \{0.25, 0.5, 1, 2, 4\}\Psi$. The first three do not meet the requirement; $t_f = \{2, 4\}\Psi$ exceed the requirement. Applying (2.8) for the constraint set optimizes this policy. The minimum time that achieves the desired performance is $t_f = 1.33\Psi$ (58min). The same analysis may be extended to optimize other policy parameters.

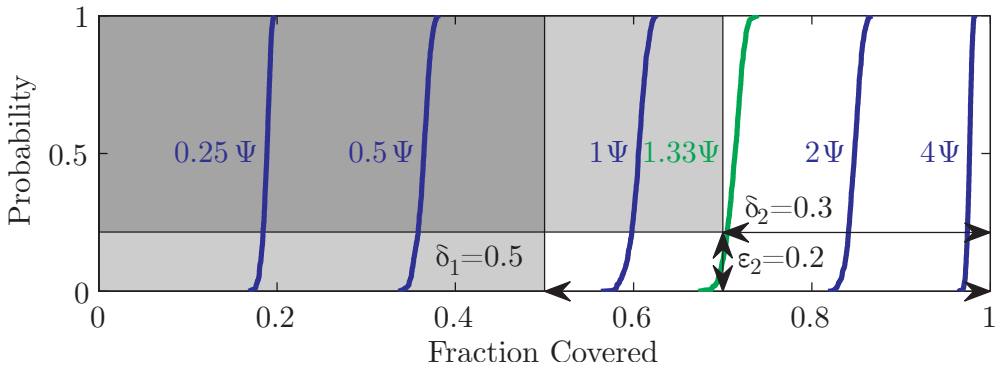


Figure 2.7: Cumulative distribution functions for execution time $t_f = \{0.25, 0.5, 1, 2, 4\}\Psi$ for a random-reflection policy under process noise $\sigma_p = 0.1$. The grey rectangles depict the desired performance constraints of $(\varepsilon_1 = 0.0, \delta_1 = 0.5) \cup (\varepsilon_2 = 0.2, \delta_2 = 0.3)$—the robot must "cover 50% or more of the free space every time and 70% or more with probability at least 80%." A search yields that $t_f = 1.33\Psi$ (58min) is the minimum time to achieve the desired performance.

Robot Sensor Selection

Sensor selection is a relevant problem in robot design. In this example, we examine the effect of increasing measurement accuracy σ_m for the boustrophedon long policy and compare it to random-reflection, independent of σ_m. Fig. 2.8 shows results for a region with width $W = 10$m, height $H = 40$m, process noise $\sigma_p = 0.1$, and measurement noise $\sigma_m = \{0.1, 0.5, 1.0, 2.5, 10\}$. We compare strategies at execution times $t_f = \{0.5, 1.5, 2.5\}\Psi$. For $t_f = \{0.5, 1.5\}\Psi$, the random strategy performs best. For $t_f = 2.5\Psi$, the boustrophedon policy with $\sigma_m = \{0.1, 0.5, 1.0, 2.5\}$ results in better performance than random-reflection (for most values of δ). These results imply that for small t_f a random strategy performs best. This is expected because random policies tend to explore more than boustrophedon policies, giving them an early advantage. This dependence is also shown in Figs. 2.2(a) and 2.6. For larger t_f decreasing values of σ_m yield better performance for the boustrophedon long policy. Modifying (2.7) to vary over candidate σ_m values—perhaps from a catalog of sensors—is straightforward. Applying (2.7) for a given execution time and fraction covered determines the optimal sensor selection. This analysis provides a tool for robot design and design trade-offs, e.g., cost and accuracy of a sensor versus time to achieve a desired performance objective.

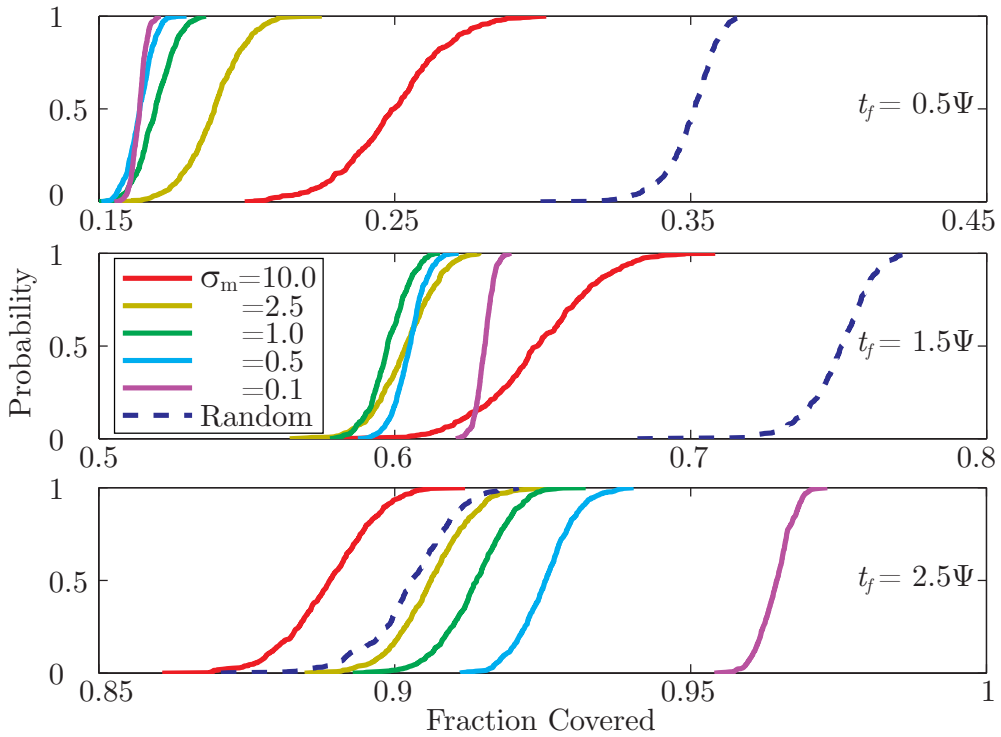


Figure 2.8: Cumulative distribution functions for execution times $t_f = \{0.5, 1.5, 2.5\}\Psi$ under process noise $\sigma_p = 0.1$ and varying measurement noise σ_m. These results are compared to a random-reflection strategy, independent of measurements. For short execution times $\{0.5, 1.5\}\Psi$, the random strategy performs best, followed by high measurement noise, and low measurement noise. As t_f increases boustrophedon paths result in better coverage—performance increases as σ_m decreases.

Chapter 3

Virtual Robot

The virtual robot is a feedback policy that achieves a desired coverage performance in a system with uncertainty in sensing and actuation. It is inspired by the result for a robot under uncertainty attempting to cover a rectangular region—that a square wave pattern with passes parallel to the short axis performs better than a square wave pattern with passes parallel to the long axis (2.2.4). The virtual robot steers a square wave path throughout the free space and is modeled as a larger, virtual robot with coverage implement size equal to the size of the square wave. See Figure 3.1.

3.1 Concept

This approach isolates local and global control policies. A global path is planned for the virtual robot and as the virtual robot tracks the global path, it generates a local path for the robot. More precisely, we use a single instance of the square wave as a motion primitive. The inputs from the virtual robot trajectory tracking controller are mapped to a set of path parameters which govern the shape of the square wave. The robot then tracks the square wave trajectory as in Figure 3.2.

The virtual robot defines a layer of abstraction for global planning. We assume that the virtual robot follows the path with negligible uncertainty and thus compute the global path using a classical approach (Section 6.3.3). Consider a region to cover and several performance objectives. The global path will remain constant for each solution, but the local policy may change to satisfy the performance objectives.

The kinematic model for the virtual robot is a functional relationship between virtual robot vehicle inputs, $\boldsymbol{u}_{vr}$—the forward speed and turn rate—and the path parameters of the square wave (Section 4.2). Bounds on the path parameters restrict the size and shape of the primitive and thus the maneuverability of the virtual robot. These bounds are specific to each system and performance objective and are computed using a method similar to that in Section 2.2.4.

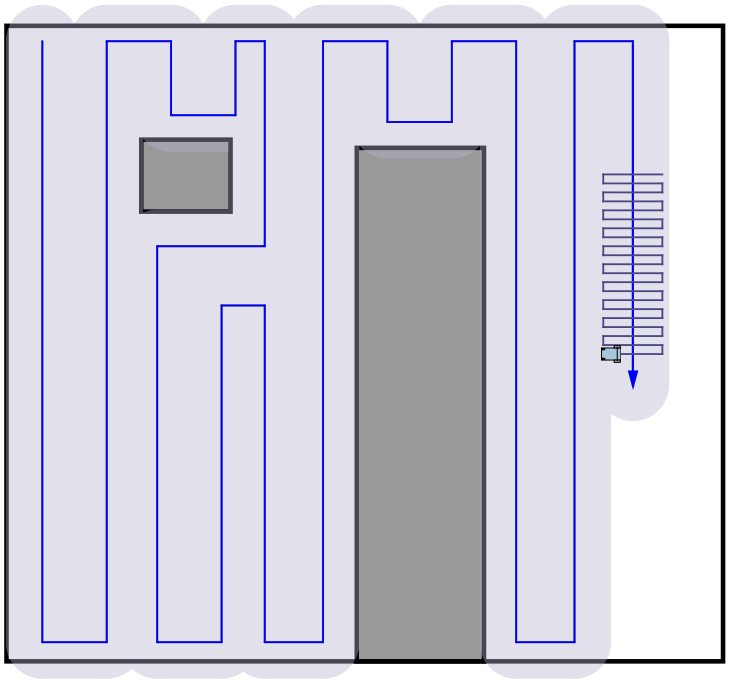


Figure 3.1: Illustration of the virtual robot strategy. A global path is planned for the virtual robot (blue), which generates a local path (square wave) for the robot. Coverage planning is simplified by separating local and global coverage plans.

The virtual robot policy is given as a block diagram in Figure 3.3. The inputs are a desired performance objective and a path. The performance objective may be in the form of a δ, ε pair or in some other form such that it corresponds to bounds on the path parameters, $\boldsymbol{p}_{max}$, and a switchback length, ℓ. The input path is a global path for the virtual robot to follow.

An LQR trajectory tracking controller tracks the reference trajectory, $\{\boldsymbol{u}^*_{vr,0}, \ldots, \boldsymbol{u}^*_{vr,n}\}$, $\{\boldsymbol{x}^*_{vr,0}, \ldots, \boldsymbol{x}^*_{vr,n}\}$, and determines a set of virtual robot vehicle inputs, $\boldsymbol{u}_{vr}$. The transformation from global to local planning occurs as $\boldsymbol{u}_{vr}$ is mapped to a set of path parameters, $\boldsymbol{p}_{vr}$. A robot reference trajectory, $\{\boldsymbol{u}^*_0, \ldots, \boldsymbol{u}^*_m\}$, $\{\boldsymbol{x}^*_0, \ldots, \boldsymbol{x}^*_m\}$, is then generated along the square wave primitive parameterized by $\boldsymbol{p}_{vr}$. For each point along the robot trajectory, a LQR trajectory tracking controller computes a set of vehicle inputs, $\boldsymbol{u}$. A lower level differential drive controller maps $\boldsymbol{u}$ to differential drive inputs, $\boldsymbol{u}_{dd}$, the right and left wheel velocities. The differential drive controller requires a set of odometric parameters, $\boldsymbol{p}$, to achieve the correct $\boldsymbol{u}$. Encoder measurements are pushed through the process model to compute a state prediction, $\hat{\boldsymbol{x}}_{dd}$. The robot updates its state estimate, $\hat{\boldsymbol{x}}$, using an EKF that fuses $\hat{\boldsymbol{x}}$ and the GPS measurement, $\hat{\boldsymbol{x}}_{GPS}$. Finally, the robot state estimate is mapped to a virtual robot state estimate, $\hat{\boldsymbol{x}}_{vr}$. We use a heuristic virtual robot state estimator in Figure 6.4.

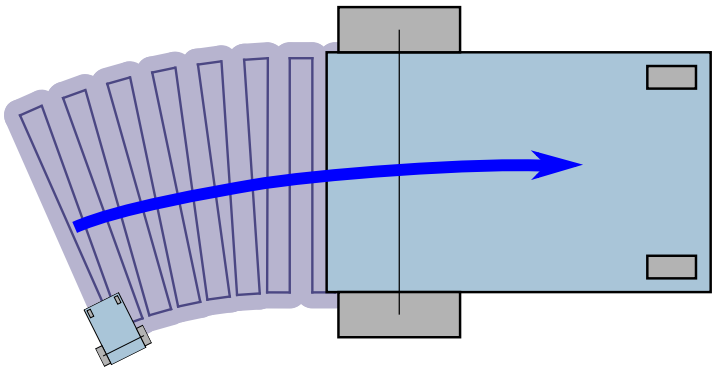


Figure 3.2: The virtual robot strategy. The virtual robot is modeled is a larger, *virtual* robot that tracks the global path (blue). The virtual robot control inputs map to a set of path parameters which govern the shape of the square wave. As the virtual robot tracks the global path, it incrementally generates a local path for the robot to track.

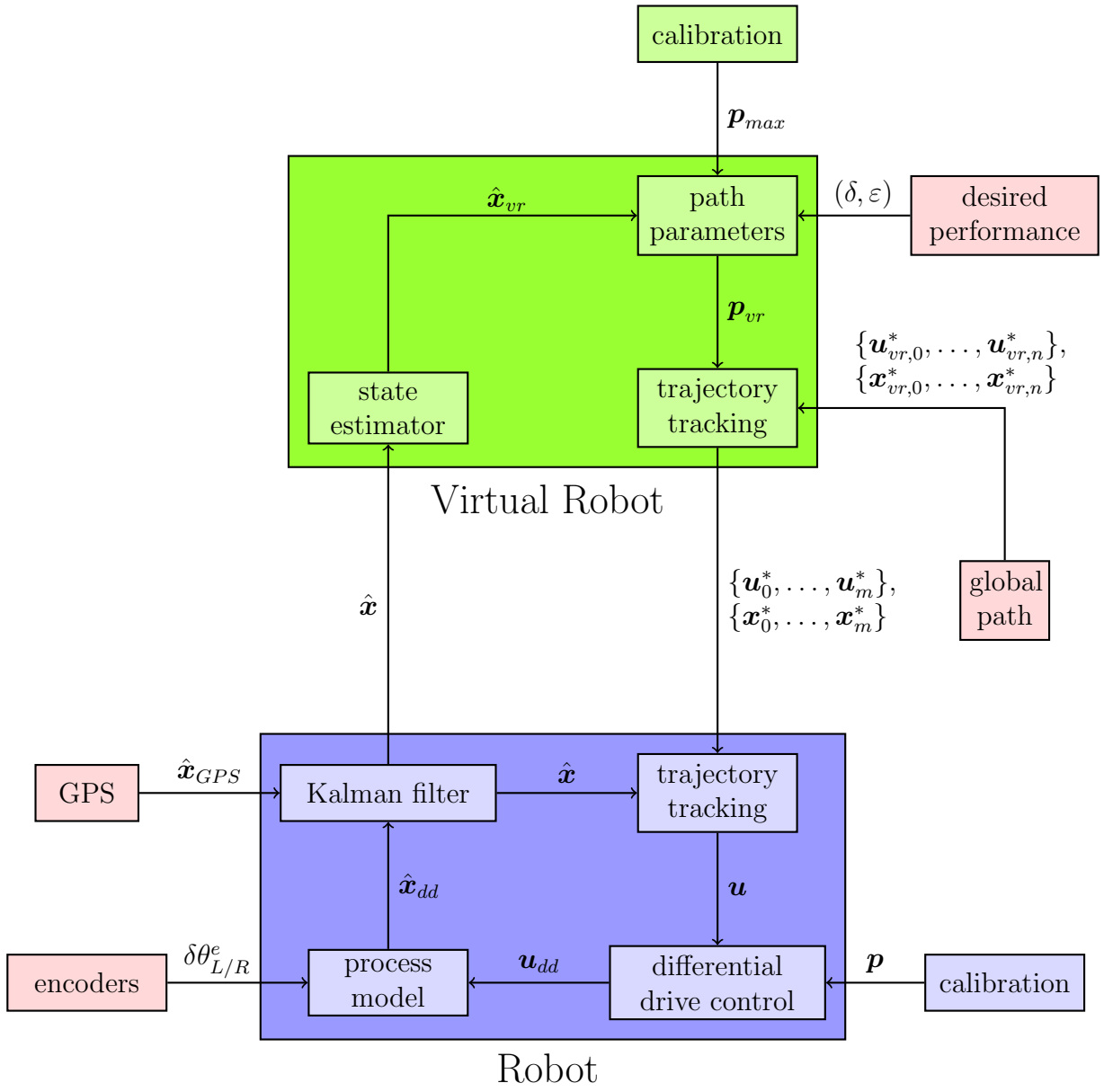


Figure 3.3: Block diagram for the virtual robot policy. Given a desired performance objective and a global path, the virtual robot incrementally generates a trajectory for the robot to follow using a square wave motion primitive. The virtual robot inputs are mapped to a set of path parameters which govern the shape of the primitive. The performance objective maps to a set of bounds on path parameters. The robot fuses both the GPS and encoder measurements in a Kalman filter for state feedback to the robot trajectory tracking controller and to the virtual robot state estimator. The differential drive controller is a lower-level controller which maintains wheel speeds to match the desired vehicle inputs.

Chapter 4

Model

4.1 Robot Kinematics

The robot is assumed to have standard differential drive equations of motion and kinematics as shown in (4.1) and (4.2). See Figure 4.1.

$$\begin{aligned} \dot{x}_1 &= v \cos x_3 \\ \dot{x}_2 &= v \sin x_3 \\ \dot{x}_3 &= \omega \end{aligned} \tag{4.1}$$

where

$$\begin{aligned} v &= \frac{p_1 u_1 + p_2 u_2}{2} \\ \omega &= \frac{p_1 u_1 - p_2 u_2}{p_3} \end{aligned} \tag{4.2}$$

where $\boldsymbol{p} = [p_1, p_2, p_3]^T$ are the odometric parameters: the right and left wheel radii and the wheel base. The state is $\boldsymbol{x} = [x_1, x_2, x_3]^T$ where x_1 and x_2 are the cartesian coordinates and x_3 is the heading. The inputs are $\boldsymbol{u} = [u_1, u_2]$ where u_1 and u_2 are the right and left wheel angular velocities. The state is $\boldsymbol{x} \in SE(2)$ or $\boldsymbol{x} \in \mathbb{R}^2 \times S^1$ and the input is $\boldsymbol{u} \in \mathbb{R}^2$. The topological space $SE(n)$, called the *special Euclidean group*, is homeomorphic to $\mathbb{R}^n \times SO(n)$.

A multitude of representations of such kinematic models appear in literature. Some are similar, but in different formats; others differ in algorithm. The main difference is typically the assumption that straight line motion and turning motion are independent. We use a form similar to the one in [46] and do not make this assumption. The time-discretized form

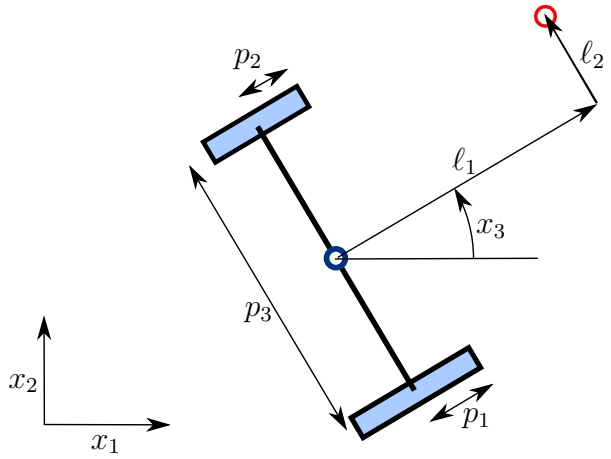


Figure 4.1: Schematic for a differential drive robot. $[p_1, p_2, p_3]$ are the right and left wheel radius and the wheel base. A sensor, drawn in red, is displaced $[\ell_1, \ell_2]$ units from the robot's center.

of (4.1) is given below.

$$\begin{bmatrix} x_{1,k+1} \\ x_{2,k+1} \\ x_{3,k+1} \end{bmatrix} = \begin{bmatrix} x_{1,k} + v_k \Delta \mathrm{t}\, \mathrm{sinc}\, \left(\frac{\omega_k \Delta \mathrm{t}}{2}\right) \cos\left(x_{3,k} + \frac{\omega_k \Delta \mathrm{t}}{2}\right) \\ x_{2,k} + v_k \Delta \mathrm{t}\, \mathrm{sinc}\, \left(\frac{\omega_k \Delta \mathrm{t}}{2}\right) \sin\left(x_{3,k} + \frac{\omega_k \Delta \mathrm{t}}{2}\right) \\ x_{3,k} + \omega_k \Delta \mathrm{t} \end{bmatrix} \tag{4.3}$$

Here, the subscripts k and $k+1$ denote the current and future time steps and Δt is the size of the time step. While [46] geometrically derives the model, we provide a mathematical derivation in A.1.

The *sinc* term is often omitted, i.e., treated as unity. As shown in [46], this is still a good approximation because the encoder sampling rate is high comapred to the wheel velocities along with the following property of *sinc* function: $(\mathrm{sinc}(x) \to 1 \mid x \to 0)$.

Process noise, or nonsystematic error, represents random errors incurred during state evolution due to unmodeled system dynamics. Variations in the terrain such as slope and friction coefficients, aging and wear of the robot, improper calibration, and robot-terrain interactions such as wheel load-up and wheel sinkage all are potential sources of random errors. We model these errors as random variables added to the inputs, i.e., the wheel turns. We assume that the errors incurred by each wheel are uncorrelated. This is reasonable because the wheels are driven by two different motors and are measured by two different optical encoders [36],[47]. We also assume the error to be zero mean and uncorrelated with the previous and next unit of travel. This is reasonable because the bias of the state evolution

is removed via calibration of the odometric parameters. We then assume that the variance of each wheel turn is proportional to the value measured by the encoder [48]. This is equivalent to [49] and [50], which model the distance traveled by each wheel instead of the change in wheel angle.

Our model of process noise is the following

$$\begin{aligned} \delta\theta_{L/R} &= \delta\theta^e_{L/R} + \nu \\ \nu &\sim \mathcal{N}\left(0, K_e^2 \left|\delta\theta^e_{L/R}\right|\right). \end{aligned} \tag{4.4}$$

Here, the input or believed wheel turns, $\delta\theta_{L/R}$, are modeled as the wheel turns measured by the encoders, $\delta\theta^e_{L/R}$, with the added random variable ν. ν is a zero-mean Gaussian random variable with variance $K_e^2 \left|\delta\theta^e_{L/R}\right|$. That is, the variance is a scale K_e^2 of the absolute values of the encoder measurements. Process noise is thus parameterized by a single value K_e with units ($\text{rad}^{1/2}$), which we have assumed to be equal for left and right wheels.

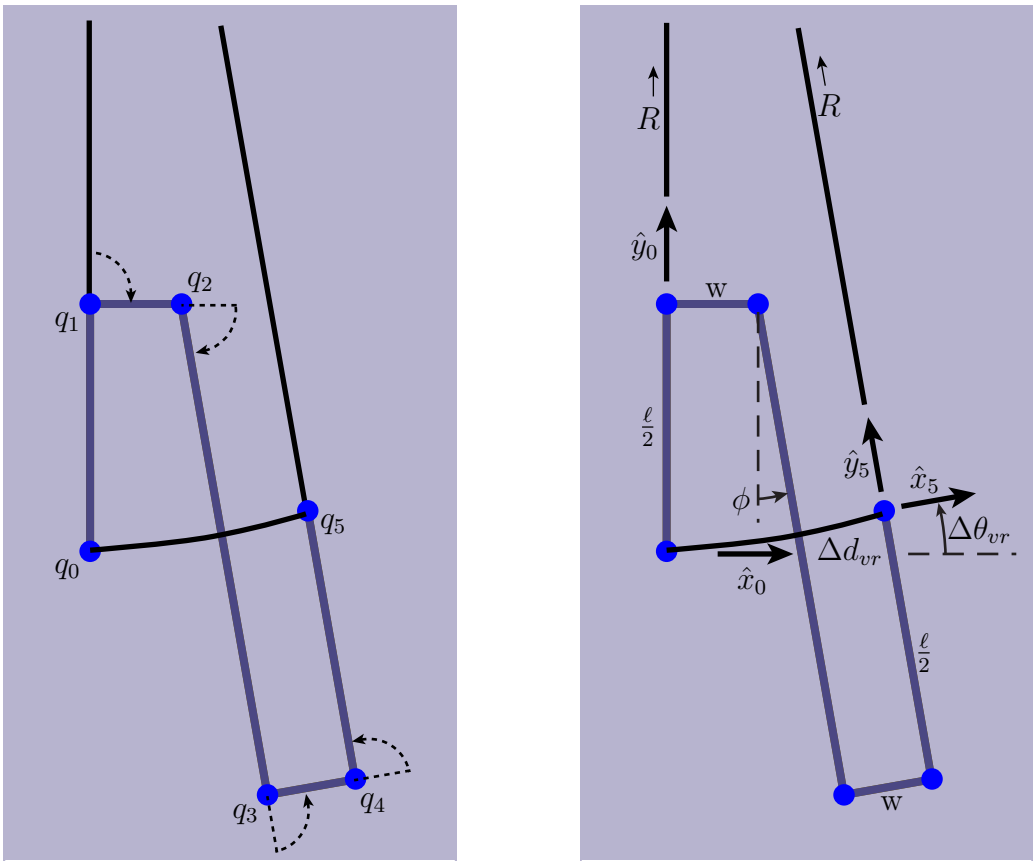
4.2 Virtual Robot Kinematics

We design the virtual robot to have similar differential equations of motion to those of the differential drive robot in (4.1). The virtual robot equations of motion are the following.

$$\begin{aligned} \dot{x}_{vr} &= v_{vr} \sin\theta_{vr} \\ \dot{y}_{vr} &= v_{vr} \cos\theta_{vr} \\ \dot{\theta}_{vr} &= \omega_{vr}. \end{aligned} \tag{4.5}$$

The virtual robot state $\boldsymbol{x}_{vr} = [x_{vr}, y_{vr}, \theta_{vr}]^T$ is in the configuration space $SE(2) = \mathbb{R}^2 \times S^1$, identical to the robot configuration space.[1] The virtual robot inputs $\boldsymbol{u}_{vr} = [v_{vr}, \omega_{vr}]$ and, as we will see, correspond to a set of input path parameters, $\boldsymbol{p}_{vr} = [\mathrm{w}, \phi]$, which belong to a bounded subset of $\mathbb{R} \times S^1$. The kinematic relationship between the virtual robot commands (forward speed and turn rate) and the path parameters are obviously very different than the kinematics of a differential drive vehicle.

[1] Here we adopt the $[x, y, \theta]$ notation instead of $[x_1, x_2, x_3]$ to simplify subscripting, though the meanings are equivalent.



(a) maneuvers and points (b) dimensions and reference frames

Figure 4.2: Geometric transformation from virtual robot vehicle inputs to robot trajectory

We then have the (simplified) discrete-time equations of motion.

$$\begin{bmatrix} x_{vr,k} \\ y_{vr,k} \\ \theta_{vr,k} \end{bmatrix} = \begin{bmatrix} x_{vr,k-1} + \Delta d_{vr,k} \cos\left(\theta_{vr,k-1} + \frac{\Delta\theta_{vr,k}}{2}\right) \\ y_{vr,k-1} + \Delta d_k \sin\left(\theta_{vr,k-1} + \frac{\Delta\theta_{vr,k}}{2}\right) \\ \theta_{vr,k-1} + \Delta\theta_{vr,k} \end{bmatrix} \tag{4.6}$$

We now analyze the kinematics of the virtual robot model, i.e., the relationship between the path parameters (ℓ, w, ϕ) and the vehicle commands (v_{vr}, ω_{vr}). We do this by explicitly computing the change in position and heading of $\boldsymbol{x}_{vr}$ over one motion primitive—a switch-back or a single instance of a square wave. These values are then divided by the time to execute the primitive, Δt_{vr}.

We begin by transforming the path spacing and overshoot from Section 2.2.4 to the length,

ℓ, and the width, w, of the square wave.

$$\ell = 2b + H$$
$$\mathrm{w} = S$$

In Figure 4.2, we compute the position of waypoints through which the robot passes. At the start of a primitive, $(x_{vr}, y_{vr}) = q_0$ and $\theta_{vr} = 0$, i.e., the virtual robot has planar configuration q_0 and heading aligned with the $\hat{x}_0$ axis. After the robot follows a single virtual robot primitive, $(x_{vr}, y_{vr}) = q_5$ and $\theta_{vr} = \Delta\theta_{vr} = \phi$. Here, $\Delta\theta_{vr}$ is the rotation between Frames 0 and 5, denoted by $(\hat{x}_0, \hat{y}_0)$ and $(\hat{x}_5, \hat{y}_5)$, respectively and we validate that this is indeed equal to ϕ.

We now show that the virtual robot distance traveled over one primitive can be evaluated as a constant curvature arc. It is first necessary to determine q_5 with respect to q_0. From Figure 4.2, the waypoints have the following values in Frame 0.

$$\begin{aligned}
q_0 &= (0, 0) \\
q_1 &= q_0 + (0, \ell/2) \\
q_2 &= q_1 + (\mathrm{w}, 0) \\
q_3 &= q_2 + (\ell \sin(\phi), -\ell \cos(\phi)) \\
q_4 &= q_3 + (\mathrm{w} \cos(\phi), \mathrm{w} \sin(\phi)) \\
q_5 &= q_4 + \left(-\frac{\ell}{2} \sin(\phi), \frac{\ell}{2} \cos(\phi) \right)
\end{aligned}$$

From above, the final position q_5 can be determined.

$$q_5 = \left(\mathrm{w} + \mathrm{w} \cos(\phi) + \frac{\ell}{2} \sin(\phi) \ , \ \frac{\ell}{2} + \mathrm{w} \sin(\phi) - \frac{\ell}{2} \cos(\phi) \right)$$

To show that for a constant ϕ, virtual robot states lie on a constant curvature path, the following condition must hold.

$$\exists R \quad \text{s.t.} \quad q_0 + R\hat{y}_0 = q_5 + R\hat{y}_5 \tag{4.7}$$

where R is a scalar. That is, R projected along $\hat{y}_0$ starting from q_0 and R projected along $\hat{y}_5$ starting from q_5, intersect. Then, we have

$$\hat{y}_5 = -\sin(\phi)\hat{x}_0 + \cos(\phi)\hat{y}_0 \ . \tag{4.8}$$

So, putting (4.7) and (4.8) together,

$$(0, R) = \left(\mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi) - R\sin(\phi) \ , \ \frac{\ell}{2} + \mathrm{w}\sin(\phi) - \frac{\ell}{2}\cos(\phi) + R\cos(\phi) \right)$$

From the $\hat{x}_1$ component above, R can be expressed as follows

$$R = \frac{\mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi)}{\sin(\phi)} \tag{4.9}$$

To verify that the constant curvature assumption holds, equate the $\hat{y}_1$ components above to get

$$\frac{\ell}{2} + \mathrm{w}\sin(\phi) - \frac{\ell}{2}\cos(\phi) = (1 - \cos(\phi))\, R$$

So substituting (4.9)

$$\frac{\ell}{2} + \mathrm{w}\sin(\phi) - \frac{\ell}{2}\cos(\phi) = (1 - \cos(\phi)) \, \frac{\mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi)}{\sin(\phi)}$$

or

$$\frac{\ell}{2}\sin(\phi) + \mathrm{w}\sin^2(\phi) - \frac{\ell}{2}\sin(\phi)\cos(\phi) = \\ \mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi) - \mathrm{w}\cos(\phi) - \mathrm{w}\cos^2(\phi) - \frac{\ell}{2}\sin(\phi)\cos(\phi)$$

which simplifies to

$$\mathrm{w}\left(\sin^2(\phi) + \cos^2(\phi)\right) = \mathrm{w}$$

and finally

$$\mathrm{w} = \mathrm{w} \quad \square$$

This validates the constant curvature assumption of the virtual robot. It is clear then that if (ℓ, w, ϕ) were held constant, the virtual robot trajectory would follow a circular path. As a check, we compute the incremental change in heading of the virtual robot, $\Delta\theta_{vr}$, over a

single motion primitive. So, we have

$$\begin{aligned}
\Delta\theta_{vr} &= \arcsin\left(\frac{p_5 \cdot \hat{x}_1}{R}\right) \\
&= \arcsin\left(\frac{\mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi)}{\frac{\mathrm{w}+\mathrm{w}\cos(\phi)+\frac{\ell}{2}\sin(\phi)}{\sin(\phi)}}\right) \\
&= \arcsin\left(\sin(\phi)\right) \\
&= \phi
\end{aligned} \tag{4.10}$$

as expected. Thus, the incremental path length of the virtual robot, Δd_{vr}, over a primitive is simply the length of the circular sector.

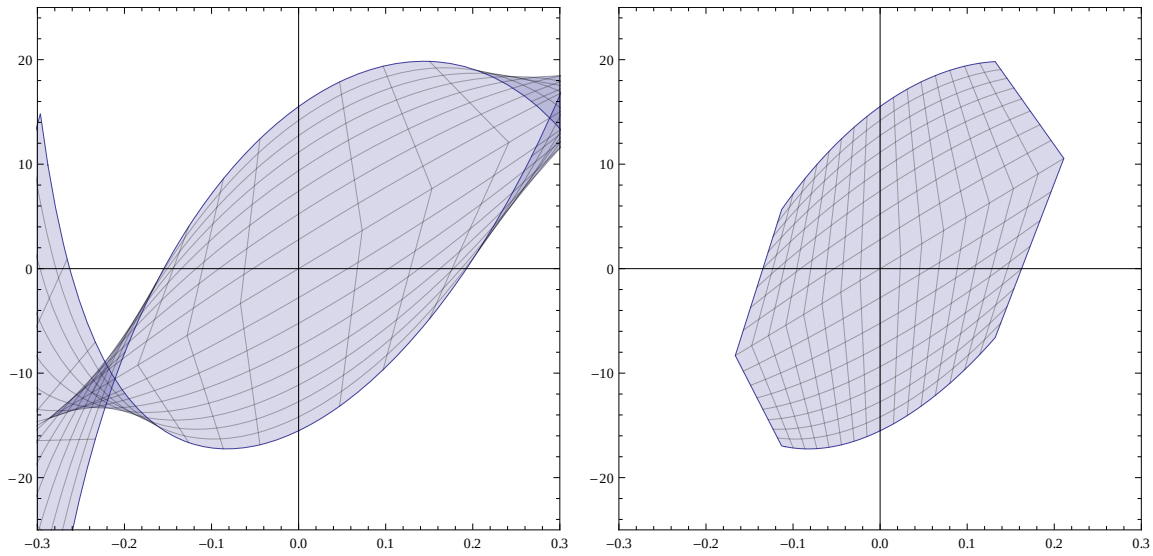
$$\begin{aligned}
\Delta d_{vr} &= R\Delta\theta_{vr} \\
&= \frac{\mathrm{w} + \mathrm{w}\cos(\phi) + \frac{\ell}{2}\sin(\phi)}{\sin(\phi)}\phi \\
&= \frac{\mathrm{w}(1+\cos(\phi))}{\sin(\phi)}\phi + \frac{\ell}{2}\phi \\
&= \frac{\mathrm{w}(1+\cos(\phi))}{\mathrm{sinc}(\phi)} + \frac{\ell}{2}\phi
\end{aligned} \tag{4.11}$$

The time over the virtual robot primitive is the sum of the lengths divided by the average robot velocity together with the sum of the angles turned divided by the average robot angular velocity.

$$\Delta t_{vr} = \frac{2\ell + 2|w|}{v} + \frac{2\pi - \phi}{\omega} \tag{4.12}$$

The absolute value of w is taken because the virtual robot has a negative velocity through a negative w. The virtual robot velocity, v_{vr} and turn rate ω_{vr} are

$$\begin{aligned}
v_{vr} &= \frac{\Delta d_{vr}}{\Delta t_{vr}} \\
\omega_{vr} &= \frac{\Delta\theta_{vr}}{\Delta t_{vr}}
\end{aligned}$$



(a) $|\phi| < 3\pi/2$ (b) $|\phi| < \pi/2$

Figure 4.3: The forward mapping from path parameters to virtual robot inputs, $\mathcal{P} \to \mathcal{U}$, for bounded values of ϕ. The x-axis is ω_{vr} and the y-axis is v_{vr}. The numerical values are not important because the inputs are functions of other parameters. Note that for large bounds on ϕ, the mapping is an injection, but not a bijection (left). For smaller values of ϕ, like the ones we consider, the mapping is a bijection (right).

where Δt_{vr} is from (4.12). So, finally

$$v_{vr} = \frac{\frac{w(1+\cos(\phi))}{\text{sinc}(\phi)} + \frac{\ell}{2}\phi}{\frac{2\ell+2|\text{w}|}{v} + \frac{2\pi+\phi}{\omega}} \tag{4.13}$$

$$\omega_{vr} = \frac{\phi}{\frac{2\ell+2|\text{w}|}{v} + \frac{2\pi+\phi}{\omega}} \tag{4.14}$$

4.2.1 Parameter Mapping

Let $\boldsymbol{p}_{vr} \in \mathcal{P}$ where $\boldsymbol{p}_{vr} = [\text{w}, \phi]$ and let $\boldsymbol{u}_{vr} \in \mathcal{U}$ where $\boldsymbol{u}_{vr} = [v_{vr}, \omega_{vr}]$. The forward mapping $\mathcal{P} \to \mathcal{U}$ is straight forward and is computed using (4.13) and (4.14). For this computation, values (ℓ, v, ω) are constant.In Figure 4.3, we map all values within the range of some $\boldsymbol{p}_{max}$ to the virtual robot inputs. This mapping is not a bijection for large values of ϕ. Thus, the inverse mapping is not as simple because it is not one to one. However, for small ϕ, that is, $\phi < \pi/2$, we show that the mapping is a bijection as in Figure 4.3.

The set of allowable $\boldsymbol{p}_{vr}$ is $\mathcal{P}_b \subset \mathcal{P}$, which is bounded by $\boldsymbol{p}_{max} = [\text{w}_{max}, \phi_{max}]$. We have heuristically chosen reasonable values such that the robot is unlikey to have an uncovered

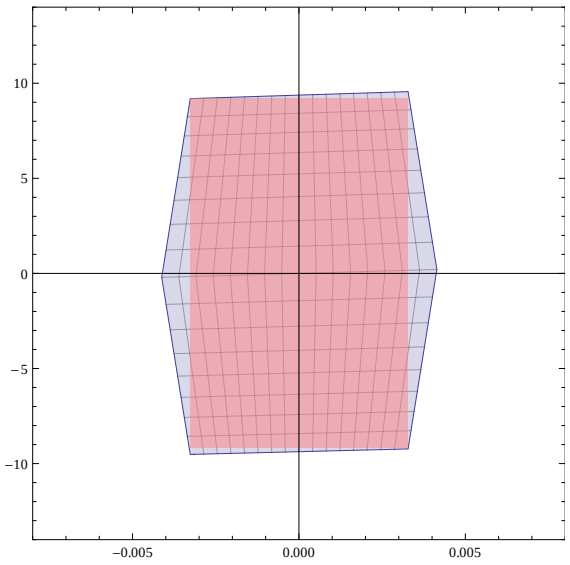


Figure 4.4: Virtual robot inputs (v_{vr}, ω_{vr}) (shaded blue) from a given set of bounds on path parameters, $\boldsymbol{p}_{max}$. We restrict the allowable set of virtual robot inputs to this region so that $\boldsymbol{p}_{max}$ is not exceeded. We refine the set to a rectangular region, $\mathcal{U}_s$ (shaded red) for simplicity.

gap between switchbacks. Thus, $\mathcal{P}_{max} \rightarrow \mathcal{U}_{max}$. The question is: when a control input $\boldsymbol{u}'_{vr} \notin \mathcal{U}_{max}$, what $\boldsymbol{p}'_{vr}$ should be used to achieve a control in $\mathcal{U}_{max}$? The idea is to do this in a simple and inexpensive way. Using the known $\mathcal{U}_{max}$ make a simple bound, e.g., a circle or a box, which can easily be applied to any $\boldsymbol{u}'_{vr} \notin \mathcal{U}_{max}$. Then, perform the inverse mapping: $\mathcal{U} \rightarrow \mathcal{P}$.

For the virtual robot, the space in which inputs are bounded, $\mathcal{U}_{max}$, is shown in Figure 4.4. From 4.3(b), it is clear that the vertices in Quadrant II or IV have the smallest magnitude of each component. The simply bounded space $\mathcal{U}_s$ is the box bounded by $\boldsymbol{p}_s \rightarrow \boldsymbol{u}_s$ in Figure 4.4, where $\boldsymbol{p}_s = [-\phi_{max}, \mathrm{w}_{max}]$.

The inverse mapping involves finding the root for the following system of nonlinear equations:

$$\frac{\frac{\mathrm{w}(1+\cos(\phi))}{\mathrm{sinc}(\phi)} + \frac{\ell}{2}\phi}{\frac{2\ell+2|\mathrm{w}|}{v} + \frac{2\pi+\phi}{\omega}} - v_{vr} = 0 \tag{4.15}$$

$$\frac{\phi}{\frac{2\ell+2|\mathrm{w}|}{v} + \frac{2\pi+\phi}{\omega}} - \omega_{vr} = 0 \tag{4.16}$$

Before using a nonlinear equation-solving algorithm, an initial guess must first be made.

This is done by approximating (4.15) as follows

$$\hat{f}_1(\mathrm{w}, \phi) = \frac{2\mathrm{w}}{\frac{2\ell+2|\mathrm{w}|}{v} + \frac{2\pi+\phi}{\omega}} \tag{4.17}$$

Equations (4.17) and (4.16) can be solved simultaneously for w and ϕ:

$$\begin{aligned} &\left(\mathrm{w} = \frac{\pi\, v\, v_{vr} + \ell\, \omega\, v_{vr}}{v\, \omega - \omega\, v_{vr} + v\, \omega_{vr}}, \phi = \frac{2\left(\pi\, v\, v_{vr}\, \omega_{vr} + \ell\, \omega\, \omega_{vr}\right)}{v\, \omega - \omega\, v_{vr} + v\, \omega_{vr}}\right) \\ &\left(\mathrm{w} = \frac{\pi\, v\, v_{vr} + \ell\, \omega\, v_{vr}}{v\, \omega + \omega\, v_{vr} + v\, \omega_{vr}}, \phi = \frac{2\left(\pi\, v\, v_{vr}\, \omega_{vr} + \ell\, \omega\, \omega_{vr}\right)}{v\, \omega + \omega\, v_{vr} + v\, \omega_{vr}}\right) \end{aligned} \tag{4.18}$$

Two sets exist because of the absolute value term. Both pairs in (4.18) are computed and substituted back into $f_1(\cdot)$ and $f_2(\cdot)$. The result with the least squared error, i.e., closest to the root, is chosen as the intitial condition, $\boldsymbol{p}_0$, for the following algorithm.

For some $\boldsymbol{u}_s \in \mathcal{U}_s$, the Newton-Raphson method finds the root, or a pair (w, ϕ) that brings $\boldsymbol{f} = [f_1(\cdot), f_2(\cdot)]^T$ to [0]. The well-known algorithm for multivariate problems is as follows

$$\boldsymbol{p}_1 = \boldsymbol{p}_0 + (-J^{-1}) \cdot \boldsymbol{f}(\boldsymbol{p}_0)$$

where J denotes the Jacobian matrix $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{p}}$ or more explicitly

$$\frac{\partial \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}}{\partial \begin{bmatrix} \mathrm{w} & \phi \end{bmatrix}}$$

Because the analytical Jacobian contains terms that divide by zero a numerical version is used—the forward difference method.

4.2.2 Simplification

While the above approach works sufficiently in simulation, on a fixed point processor, it is desirable to minimize the number of floating point calculations because the numerical errors propagate. To simplify, we use the following approximation.

$$\Delta t_{vr,k} = \frac{2\mathrm{w}}{\frac{2\ell+2|\mathrm{w}|}{v} + \frac{2\pi+\phi}{\omega}} = \Delta t^*_{vr,k-1} \tag{4.19}$$

Here, we approximate the duration of the current virtual robot segment, $\Delta t_{vr,k}$, as the previous target duration $\Delta t_{vr,k-1}$. This reduces the system of equations in (4.15) and (4.16) to

$$\begin{aligned} \frac{\frac{\mathrm{w}(1+\cos(\phi))}{\mathrm{sinc}(\phi)} + \frac{\ell}{2}\phi}{\Delta t_{vr}} - v_{vr} &= 0 \\ \frac{\phi}{\Delta t_{vr}} - \omega_{vr} &= 0 \end{aligned} \tag{4.20}$$

This is solved in closed form in (4.21) below.

$$\begin{aligned} w &= \frac{\Delta t_{vr} \operatorname{sinc}\left(\Delta t_{vr}\omega_{vr}\right)\left(2v_{vr} - \ell\omega_{vr}\right)}{2\left(1+\cos\left(\Delta t_{vr}\omega_{vr}\right)\right)} \\ \phi &= \Delta t_{vr}\omega_{vr} \end{aligned} \tag{4.21}$$

We verify that the simplificiation is sufficient and run the virtual robot using the Newton-Raphson method and using the simplified method 20 times each.

4.3 Sensors

The robot is modeled with the following three sensor types.

1. a locally referenced sensor, e.g., wheel encoders
2. a globally referenced sensor, e.g., GPS
3. a boundary sensor, e.g., a hall-effect sensor

Throughout this work, we use "locally referenced sensor" and "wheel encoders" or simply "encoders" interchangebly, as well as "globally referenced sensor" and "GPS." As stated earlier, wheel encoders report individual wheel turn angles, which from (4.1) and (4.2) can be integrated to infer full state information. The GPS sensor returns a full state estimate. We assume that the GPS sensor is either part of an inertial measurement unit which measures heading or has some low-level computation to infer heading.

The boundary or proximity sensor able to detect boundary crossings is mounted a known distance ℓ_1 in front of the center of the wheelbase and ℓ_2 along the wheelbase as depicted in Figure 4.1. The sensor location in the x_2-direction is thus

$$y = x_2 + \ell_1 \sin x_3 + \ell_2 \cos x_3 \tag{4.22}$$

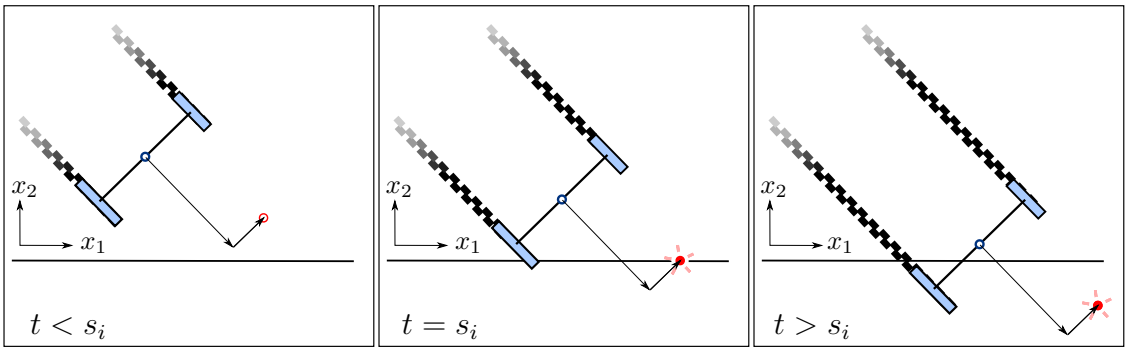


Figure 4.5: The boundary line is drawn in black. The sensor returns 1 when outside the boundary and 0 otherwise.

We consider a boundary sensor with a binary output: 0 if inside and 1 if outside the boundary. That is, for the simplified world in Figure 4.5 where the boundary is a line along $x_2 = 0$,

$$\begin{cases} 0, & \text{if } y > 0 \\ 1, & \text{otherwise} \end{cases}$$

The odometry error covariance matrix $\boldsymbol{R}_k$ describes the uncertainty in the motion model. From our model of process noise and from [49] and [50], we have the covariance matrix for straight line (SL) motion, Cov_{SL}, and for zero-point turns (ZPT), Cov_{ZPT}. For clarity, $\boldsymbol{R}_k$ adopts either Cov_{SL} or Cov_{ZPT} when performing a straight line or zero-point turn, respectively. The approximation of Cov_{SL} assumes that the left and right wheels rotate the same amount and that the heading is constant. We are interested in the motion error between the current state and the previous state. By assuming that the initial state is accurately known and is $[0\ 0\ 0]^T$, we have

$$\text{Cov}_{SL} = \begin{bmatrix} \frac{\Delta d}{2} K_e^2 & 0 & 0 \\ 0 & \frac{2\Delta d^3}{3p_3^2} K_e^2 & \frac{\Delta d^2}{p_3^2} K_e^2 \\ 0 & \frac{\Delta d^2}{p_3^2} K_e^2 & \frac{2\Delta d}{p_3^2} K_e^2 \end{bmatrix} \tag{4.23}$$

where Δd is the believed incremental distance traveled and K_e is the nonsystematic error parameter from (4.4).

The error covariance matrix for ZPTs assumes that both wheels rotate by the same amount but in opposite directions. Again, we are interested in the motion error and make the same

assumption as above, and have

$$\begin{aligned}
\mathrm{Cov}_{ZPT}(1,1) &= \frac{p_3}{16}\left[2\Delta\theta + \sin\left(2\Delta\theta\right)\right]K_e^2 + \mathrm{sgn}(\Delta\theta) \\
\mathrm{Cov}_{ZPT}(2,2) &= \frac{p_3}{16}\left[2\Delta\theta - \sin\left(2\Delta\theta\right)\right]K_e^2 + \mathrm{sgn}(\Delta\theta) \\
\mathrm{Cov}_{ZPT}(3,3) &= \frac{\Delta\theta}{p_3}K_e^2 \\
\mathrm{Cov}_{ZPT}(1,2) &= \mathrm{Cov}_{ZPT}(2,1) = \frac{p_3}{16}\left[1 - \cos(2\Delta\theta)\right]K_e^2\,\mathrm{sgn}(\Delta\theta) \\
\mathrm{Cov}_{ZPT}(1,3) &= \mathrm{Cov}_{ZPT}(3,1) = 0 \\
\mathrm{Cov}_{ZPT}(2,3) &= \mathrm{Cov}_{ZPT}(3,2) = 0
\end{aligned} \tag{4.24}$$

where $\Delta\theta$ is the believed incremental change in heading and p_3 is the wheel separation. In simulation, both Cov_{SL} and Cov_{ZPT} are known because the injected process noise is known, i.e., we know K_e. In a real system, however, K_e must be inferred from experimental data. We do not provide a method for explicitly determining K_e, but provide two heuristic methods of computing the odometry error covariance matrix in A.2 and A.3. For a method to compute K_e, refer to [48].

The GPS measurement model h assumes a full state measurement. Thus, h is equivalent to the linearized measurement model $\boldsymbol{H}_k$, which is the identity.

$$\boldsymbol{H}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Chapter 5

Calibration

Autonomous mobile robots rely on a system model and sensor information for control and location estimation. Calibration, a form of system identification, estimates the values of the systematic parameters preventing systematic errors from dominating this estimation. Specifically, errors in the localization from odometry information, i.e., dead reckoning, were shown to have a square dependence on distance traveled [51] for systematic parameters, thus making calibration a useful and necessary tool.

Calibration is typically performed when the robot is not in operation—called offline calibration. Most robots undergo offline calibration prior to usage and it is common for robots to require recalibration due to a change in systematic parameters over time. For example, the wheels of a robot wear down and decrease in size with usage. We present several well-known methods of offline calibration such as the UMBmark test [36] and the EKF.

For systems with continuous full state observations, Antonelli and Chiaverini [52] use two successive least squares approximations. Partial state measurements can be handled by the EKF, but for specific measurement models [53], [54], [48]. Online calibration is of particular interest when modeling the effective physical parameters under different surface conditions and as they warp over time. The EKF can also be implemented online [48].

5.1 UMBmark Test

The UMBmark test aims to identify the odometric parameters (r_R, r_L, d)—the right wheel diameter, left wheel diameter, and wheelbase, respectively—based on absolute position measurements after the execution of predefined trajectories. It attempts to minimize nonsystematic errors by running the robot on a smooth surface, i.e. non-slip, effectively isolating the vehicle's exhibition of systematic errors. While not all inclusive, Borenstein and Feng observed that the two most significant sources of systematic error are the following [36]:

1. unequal wheel diamters
    - rubber tires are often not manufactured with identical diameters

- each tire will compress differently under asymetric weight distribution

2. uncertainty about the effective wheelbase

   - rubber tires contact a surface at an *area*
   - friction effects increase the effective wheel base

and thus created a procedure for identifying these calibration factors. The common causes of these uncertainties is also given above.

5.1.1 Method

Borenstein and Feng recommend setting up perpendicular walls to act as an intertial reference frame and to aid in repeatability [36]. The next step is to ensure that the actual average of the left and right wheel diameters is known; if unchecked, the vehicle will experience an additional dead-reckoning error, which is referrered to as the *scaling error*, E_s.[1] We maintain consistent notation with [36] throughout this section. The robot is programmed to move straight forward for a distance, e.g., 3 m. (A longer distance will produce a more accurate result.) The actual length between starting and stopping positions, L_{actual}, is compared to the length as computed from the robot, L_{calc}. This measurement can be done with a tape measure, but must be with an accuracy of 0.03% of L_{calc}, e.g. 1 mm in this case. The scaling error is defined in (5.1) and is slightly modified from [36] for clarity. The compensated nominal wheel diameter, for input into the dead-reckoning program, is shown in (5.2).

$$E_s = \frac{L_{actual}}{L_{calc}} \tag{5.1}$$

$$D_n^* = E_s D_n \tag{5.2}$$

where D_n is the original, nominal average wheel diameter and D_n^* is the new, measured average wheel diameter.

The robot is programmed to travel a square path of side length D, n times, both clockwise (CW) and counter-clockwise (CCW). The starting and stopping positions should be measured with respect to some inertial reference frame such that the robot begins movement in the +x-direction. The desired initial position and trajectory is shown in Figure 5.1. The offsets of the Cartesian coordinates of the final position from the initial position are calculated for each run. These errors are shown in (5.3) and (5.4) and are calculated for both CW and

[1] The remainder of the test assumes the scaling error has been corrected for.

CCW directions.

$$e_x = x_{abs} - x_{calc} \tag{5.3}$$

$$e_y = y_{abs} - y_{calc} \tag{5.4}$$

The sequence of computations in (5.5)-(5.12) produces the correction factors for the left wheel, right wheel, and wheelbase (c_l, c_R, c_d), which, if multiplied by the original values, gives the estimated true values.

$$x_{c.g.,CW/CCW} = \frac{1}{n}\sum_{i=1}^{n} e_{x,i,CW/CCW} \tag{5.5}$$

$$y_{c.g.,CW/CCW} = \frac{1}{n}\sum_{i=1}^{n} e_{y,i,CW/CCW} \tag{5.6}$$

$$\alpha = \text{avg}\left(\frac{x_{c.g.,CW} + x_{c.g.,CCW}}{-4D}, \frac{y_{c.g.,CW} - y_{c.g.,CCW}}{-4D}\right) \tag{5.7}$$

$$\beta = \text{avg}\left(\frac{x_{c.g.,CW} - x_{c.g.,CCW}}{-4D}, \frac{y_{c.g.,CW} + y_{c.g.,CCW}}{-4D}\right) \tag{5.8}$$

$$E_d = \frac{D + B\sin(\beta/2)}{D - B\sin(\beta/2)} \tag{5.9}$$

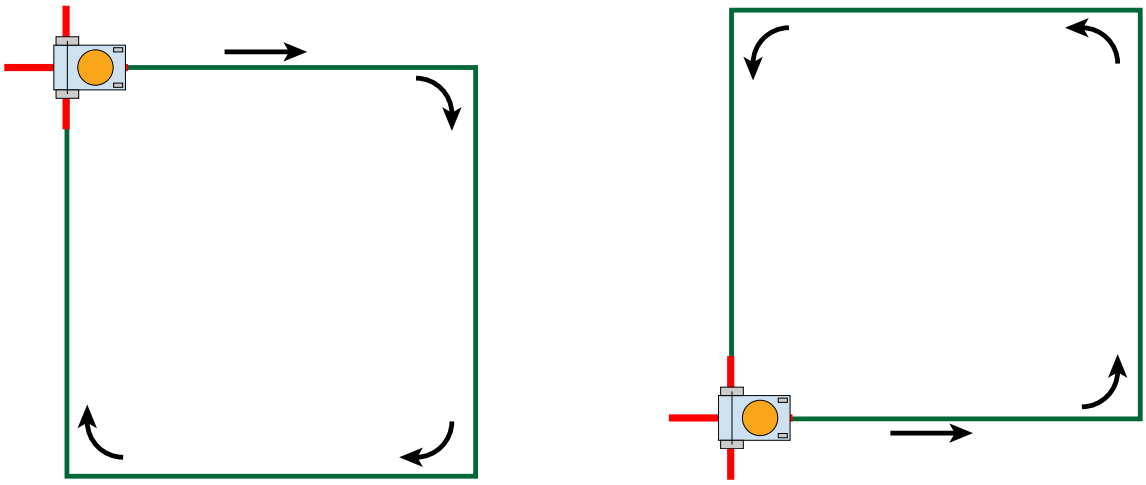
$$c_b = \frac{\pi}{\pi - \alpha} \tag{5.10}$$

$$c_l = \frac{2}{E_d + 1} \tag{5.11}$$

$$c_r = E_d c_l \tag{5.12}$$

For a detailed derivation see [36] (pp. 11, 29-36).

While extra care must be taken for the high measurement accuracy required by this test, it is still a low demand for the information it produces. However, the UMBark test has several drawbacks that are inherent in the once-per-test measurement style and due to some simplifying assumptions. Systematic errors are vehicle specific and are treated as constant during runs. This is valid seeing as wheels typically do not degrade during the time span required by the UMBmark test. Over time, though, odometric parameters will warp or degrade, thus requiring recalibration. Also, the robot will likely experience *some* amount of wheel slippage due to imperfections in the surface, tire, or control algorithm, which will be absorbed into the correction factors, thereby inducing some deviation in the calibration.



(a) Clockwise (CW)

(b) Counter Clockwise (CCW)

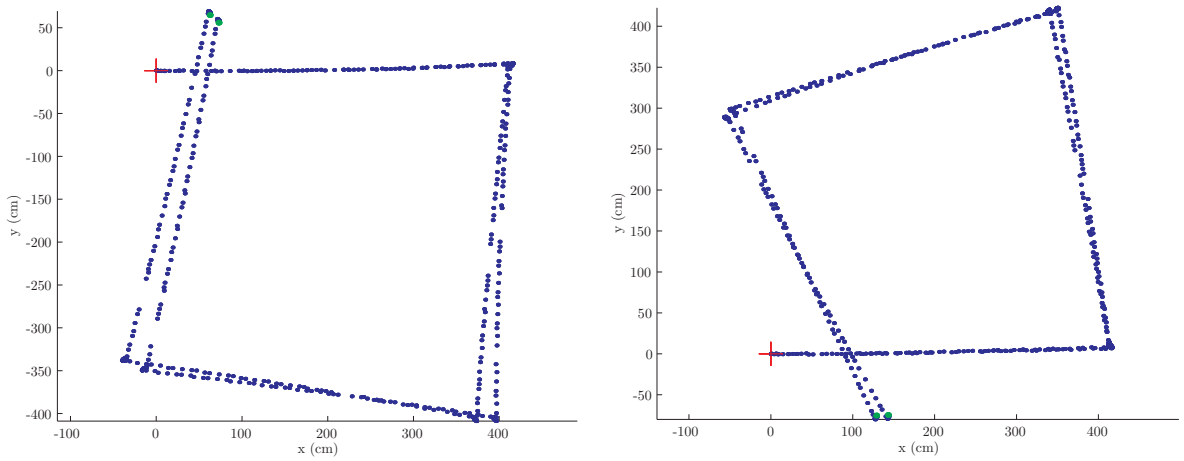
Figure 5.1: The UMBmark Bi-Directional Square Test [36]. The desired initial and final positions are the same - marked by the robot. The square trajectories of side length D are in green with arrows indicating direction.

5.1.2 Implementation

The UMBmark calibration was implemented on a mobile robot for a side length of 4 m. The control algorithm was a simple ON/OFF command for each wheel. In the future, a smarter controller such as the one in Section B.3 would produce better results, by reducing wheel slip. Due to time constraints, only two successful tests were run for each CW and CCW directions. These trajectories are shown in Figures 5.2(a) and 5.2(b), respectively. The results from applying the above algorithm to this data is shown in Table 5.1.

	r_R	r_L	d
Initial	22.0	22.0	44.5
Calibrated	22.772	22.705	44.017

Table 5.1: Odometric parameters in units of centimeters before (initial) and after (calibrated) UMBmark calibration.



(a) CW (b) CCW

Figure 5.2: The UMBmark Bi-Directional Square Test [36]. Ground truth measurements of a mobile robot programmed to execute the square trajectories in Figure 5.1. The poor trajectory shape indicates miscalibration. The red cross is the initial position.

5.2 EKF

A Kalman filter is the most common way to perform state estimation because of its ability to fuse the current state estimate with measurements from multiple sensors. It uses a system model and measurements from internal and exteroceptive sensors to maintain an estimate of both the robot pose and the covariance matrix describing the uncertainty of that estimate. By combining previous estimates and incoming measurements, a Kalman filter minimizes the variance of the resulting estimate. Because the Kalman filter requires a linear system, the EKF, which uses linearized state transition and measurement matrices, is applied here to handle the non-linear motion model in (5.13). Here, we use a laser range finder as an external measurement source.

5.2.1 Motion Model

It is known that the state $\boldsymbol{x}_k$ can be written in the form of a state transition model

$$\boldsymbol{x}_k = g\left(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k\right)$$

Here we choose the simplified form of the motion model as in Section 4.1, i.e. $sinc \rightarrow 1$, but with a slightly different form. This is to simplify the Kalman filtering equations and

to maintain consistency with [55] (pp. 45-56). Additionally, we have changed $v_k\Delta\text{t}$ to Δd_k and $\omega_k\Delta\text{t}$ to $\Delta\theta_k$ where these are defined in (5.14) and (5.15). This is because the encoders directly measure wheel angle.[2]

$$\begin{bmatrix} x_{1,k} \\ x_{2,k} \\ x_{3,k} \end{bmatrix} = \begin{bmatrix} x_{1,k-1} + \Delta d_k \cos\left(x_{3,k-1} + \frac{\Delta\theta_k}{2}\right) \\ x_{2,k-1} + \Delta d_k \sin\left(x_{3,k-1} + \frac{\Delta\theta_k}{2}\right) \\ x_{3,k-1} + \Delta\theta_k \end{bmatrix} \tag{5.13}$$

From [55], the true parameters p_1, p_2, and p_3 are approximated by modifying the nominal values p_1^*, p_2^*, and p_3^* where the quantities δ_1, δ_2, and δ_3 are proportional modifiers.

$$\begin{aligned} p_1 &= \delta_1 p_1^* \\ p_2 &= \delta_2 p_2^* \\ p_3 &= \delta_3 p_3^* \end{aligned}$$

The incremental distance and rotation for the motion model become

$$\Delta d_k = \frac{\delta_1 p_1^* \alpha_R + \delta_2 p_2^* \alpha_L}{2} \tag{5.14}$$

$$\Delta\theta_k = \frac{\delta_1 r_1^* \alpha_R - \delta_2 p_2^* \alpha_L}{\delta_3 p_3^*} \tag{5.15}$$

where the system input is

$$\boldsymbol{u}_k = [\alpha_R, \alpha_L].$$

Here α_L and α_R are the left and right incremental wheel turns as sampled by the encoders (converted to radians). The state vector can then be augmented with the three unknown parameters as in (5.16) so that both the state and odometric parameters can be estimated.

$$\boldsymbol{x}_k = [x\ y\ \theta\ \delta_1\ \delta_2\ \delta_3]_k^T \tag{5.16}$$

It is implicit here that δ_1, δ_2, and δ_3 are updated by the Kalman gain as the reader will see in (5.25). These values are simply reassigned each iteration. So the three additional state

[2] In order to obtain wheel velocities, we would need to take, for example, a first order discrete derivative. Then, this would be integrated. We directly measure wheel angular positions to avoid the numerical error incurred by these operations.

transition functions are

$$\delta_{1,k} = \delta_{1,k-1} \tag{5.17}$$

$$\delta_{2,k} = \delta_{2,k-1} \tag{5.18}$$

$$\delta_{3,k} = \delta_{3,k-1} \tag{5.19}$$

5.2.2 EKF Model

The state transition probability in (5.20) is the nonlinear function g with uncertainty ε_k, which has zero mean and a covariance denoted by R_k; g is the nonlinear function (5.13). The measurement probability in (5.21) is the nonlinear function h with uncertainty δ_t, which has zero mean and a covariance denoted by Q_k.

$$\boldsymbol{x}_k = g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \varepsilon_k \tag{5.20}$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \delta_k \tag{5.21}$$

The standard equations for the extended Kalman filter from [56] can be applied to the system model in (5.20) and (5.21) are given below:

$$\overline{\boldsymbol{x}}_k = g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{5.22}$$

$$\overline{\boldsymbol{\Sigma}}_k = \boldsymbol{G}_k \boldsymbol{\Sigma}_{k-1} \boldsymbol{G}_k^T + \boldsymbol{R}_k \tag{5.23}$$

$$\boldsymbol{K}_k = \overline{\boldsymbol{\Sigma}}_k \boldsymbol{H}_k^T \left(\boldsymbol{H}_k \overline{\boldsymbol{\Sigma}}_k \boldsymbol{H}_k^T + \boldsymbol{Q}_k \right)^{-1} \tag{5.24}$$

$$\boldsymbol{x}_k = \overline{\boldsymbol{x}}_k + \boldsymbol{K}_k \left(\boldsymbol{z}_k - \boldsymbol{H}_k \overline{\boldsymbol{x}}_k \right) \tag{5.25}$$

$$\boldsymbol{\Sigma}_k = \left(I - \boldsymbol{K}_k \boldsymbol{H}_k \right) \overline{\boldsymbol{\Sigma}}_k \tag{5.26}$$

The prediction step of the filter consists of (5.22) and (5.23), denoted by the bar, and the correction step consists of (5.25) and (5.26).

5.2.3 EKF Terms

Process Model

$\boldsymbol{G}_k$ is the linearized system matrix and is defined as the slope of g in (5.27). From a total of 6 equations: three in (5.13) and (5.17) through (5.19) and a 6-component state vector, $\boldsymbol{G}_k$

becomes a (6×6) Jacobian matrix and is broken up into four (3×3) matrices in (5.28).

$$\boldsymbol{G}_k = g'(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \boldsymbol{x}_{k-1}} \tag{5.27}$$

$$= \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{F}_k \\ 0 & I \end{bmatrix} \tag{5.28}$$

where

$$\boldsymbol{A}_k = \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{k-1}}$$

$$= \begin{bmatrix} 1 & 0 & -\Delta d_k \sin\phi \\ 0 & 1 & -\Delta d_k \cos\phi \\ 0 & 0 & 1 \end{bmatrix} \tag{5.29}$$

and where

$$\boldsymbol{F}_k = \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix}_{k-1}}$$

$$f_{11} = \cos(\phi)\frac{\partial \Delta d_k}{\partial \delta_1} - \frac{1}{2}\sin(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_1}$$
$$f_{12} = \cos(\phi)\frac{\partial \Delta d_k}{\partial \delta_2} - \frac{1}{2}\sin(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_2}$$
$$f_{13} = -\frac{1}{2}\sin(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_3}$$
$$f_{21} = \sin(\phi)\frac{\partial \Delta d_k}{\partial \delta_1} + \frac{1}{2}\cos(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_1}$$
$$f_{22} = \sin(\phi)\frac{\partial \Delta d_k}{\partial \delta_2} + \frac{1}{2}\cos(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_2}$$
$$f_{23} = \frac{1}{2}\cos(\phi)\Delta d_k \frac{\partial \Delta \theta_k}{\partial \delta_3}$$
$$f_{31} = \frac{\partial \Delta \theta_k}{\partial \delta_1}$$
$$f_{32} = \frac{\partial \Delta \theta_k}{\partial \delta_2}$$
$$f_{33} = -\frac{\partial \Delta \theta_k}{\partial \delta_3}$$

The corresponding derivatives of $\Delta d_k(\delta_1, \delta_2)$ and $\Delta \theta_k(\delta_1, \delta_2, \delta_3)$ with respect to the augmented state parameters are as follows.

$$\frac{\partial \Delta d_k}{\partial \delta_1} = \frac{p_1 \alpha_R}{2}$$
$$\frac{\partial \Delta d_k}{\partial \delta_2} = \frac{p_2 \alpha_L}{2}$$
$$\frac{\partial \Delta \theta_k}{\partial \delta_1} = \frac{p_1 \alpha_R}{p_3 \delta_3}$$
$$\frac{\partial \Delta \theta_k}{\partial \delta_2} = -\frac{p_2 \alpha_L}{p_3 \delta_3}$$
$$\frac{\partial \Delta \theta_k}{\partial \delta_3} = \Delta \theta_k \left(-\frac{1}{\delta_3}\right)$$

where $\phi = \left(\theta_{k-1} + \frac{\Delta \theta_k}{2}\right)$. Matrix $\boldsymbol{A}_k$ has a clean result. However, $\boldsymbol{F}_k$ is difficult to simplify. Thus, the symbolic computations were performed in two steps:

1. the partial derivatives of g with respect to the augmented state parameters treating Δd_k and $\Delta\theta_k$ as functions, in this case: $\Delta d_k(\delta_1, \delta_2)$ and $\Delta\theta_k(\delta_1, \delta_2, \delta_3)$

2. the partial derivatives of $\Delta d_k(\delta_1, \delta_2)$ and $\Delta\theta_k(\delta_1, \delta_2, \delta_3)$ with respect to the state parameters

where g is the nonlinear function that models the effect that inputs have on the state in (5.13). Another reason for decomposing this computation is to make easier the use of a different motion model, particularly one including the *sinc* term.

The (6×6) process noise matrix $\boldsymbol{R}_k$ describes the uncertainty in the motion model and is broken up into four (3×3) matrices in (5.30).

$$\boldsymbol{R}_k = \begin{bmatrix} \boldsymbol{N}_k & 0 \\ 0 & \boldsymbol{S} \end{bmatrix} \tag{5.30}$$

This is normally computed given a process noise model as given in Section 4.3. If neither of these methods are feasible, another method from [55] is presented[3].

$$\begin{aligned} \boldsymbol{N}_k &= \boldsymbol{P}_k \boldsymbol{N}_{max} \boldsymbol{P}_k^T \\ \boldsymbol{N}_k &= (3 \times 2)(2 \times 2)(2 \times 3) = (3 \times 3) \end{aligned} \tag{5.31}$$

where $\boldsymbol{P}_k$ is given by

$$\begin{aligned} \boldsymbol{P}_k &= \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \boldsymbol{u}_k} \\ \boldsymbol{P}_k &= \begin{bmatrix} \cos\phi & -\frac{1}{2}\Delta d_k^* \sin\phi \\ \sin\phi & \frac{1}{2}\Delta d_k^* \cos\phi \\ 0 & 1 \end{bmatrix}. \end{aligned}$$

The work in [55] presents two options for determining $\boldsymbol{N}_{max}$: "Gaussian white noise processes" and "propagating modeling uncertainty". The latter has been chosen here for the following reasons:

- dependence on parameter uncertainties: $\tilde{\delta}p_1$, $\tilde{\delta}p_2$, and $\tilde{\delta}p_3$ (explained below)
- dependence on robot movement
  - no movement $\rightarrow$ no increase in estimation error

[3] Larsen's notation of $\boldsymbol{G}_k$, $\boldsymbol{Q}_k$, and $\boldsymbol{Q}_{max}$ have been replaced here with $\boldsymbol{P}_k$, $\boldsymbol{N}_k$, and $\boldsymbol{N}_{max}$ to prevent confusion

- easier to scale optimally [55]

The "propogating modeling uncertainty" covariance matrix assumes that the odometric parameters are known with some uncertainty (denoted by *):

$$\begin{aligned} p_1^* &= p_1 + \tilde{\delta} p_1 \\ p_2^* &= p_2 + \tilde{\delta} p_2 \\ p_3^* &= p_3 + \tilde{\delta} p_3 \end{aligned} \tag{5.32}$$

The values of $\tilde{\delta} p_1$, $\tilde{\delta} p_2$, and $\tilde{\delta} p_3$ can be based on manufacturer specifications, estimated fluctuation, or estimated measuring error. The effect from these uncertain odometric parameters on Δd_k and $\Delta \theta_k$ is evaluated. The difference between the maximum and minimum deviations yields the input vector uncertainties $\delta \Delta d$ and $\delta \Delta \theta$ below.

$$\delta \Delta d_k = |\tilde{\delta} p_1 \alpha_R| + |\tilde{\delta} p_2 \alpha_L| \tag{5.33}$$

$$\delta \Delta \theta_k = \frac{2 p_3}{p_3^2 - \Delta p_3^2} \left(|\tilde{\delta} p_3 \Delta \theta_k| + |\tilde{\delta} p_1 \alpha_R| + |\tilde{\delta} p_2 \alpha_L| \right) \tag{5.34}$$

A derivation of these equations can be found in Appendix A.2. The matrix describing squared uncertainties of Δd_k^* and $\Delta \theta_k^*$ for substitution into (5.31) is

$$N_{max} = \begin{bmatrix} (\delta \Delta d_k)^2 & 0 \\ 0 & (\delta \Delta \theta_k)^2 \end{bmatrix} \tag{5.35}$$

where $\delta \Delta d_k$ and $\delta \Delta \theta_k$ can be found from (5.33) and (5.34), respectively.

S is a fictitious noise injection on the augmented states that is assumed constant; this allows for fluctuation from the initial assignments. Setting S as a diagonal matrix with eigenvalues that are small enough to reduce noise and large enough to ensure convergence will prevent converging to false estimates [55] (p.57).

Measurement Model

The partial state measurement from the laser range finder is $\boldsymbol{z}_k = [x_{1,k}\ x_{2,k}]^T$. Because the external sensor directly measures x_1 and x_2, h is equivalent to its linearized form, $\boldsymbol{H}_k$.

$$\boldsymbol{H}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

This is a different form than in Section 4.3 because the laser range finder measures only the planar coordintes and not the heading. However, if using a sufficiently accurate GPS sensor, we use the measurement model from Section 4.3.

$\boldsymbol{Q}_k$ is a (2×2) covariance matrix describing the measurement uncertainty. The laser range finder specifies that it is accurate within 3 mm. However, the uncerainty used was much higher for the following two reasons:

1. It is instructed in [48] to overestimate the variances for both the motion model and the measurement.

2. The prism, the location from which measurements are taken, is mounted approximately one foot above the wheel axel causing differences in elevation at the left and right wheels to directly affect the position measurement.

$\boldsymbol{Q}_k$ is defined below.

$$\boldsymbol{Q}_k = \boldsymbol{Q}_{mag} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{5.36}$$

where $\boldsymbol{Q}_{mag}$ was 30 times the manufacturer-specified error. If using a sensor with a full-state measurement, $\boldsymbol{Q}_k$ is a (3×3) covariance matrix.

Matrix Dimensions

For clarity, after defining the above terms, (5.22)–(5.26) have the matrix dimensions below.

$$\begin{aligned}
(6 \times 1) &= (6 \times 1) \\
(6 \times 6) &= (6 \times 6)(6 \times 6)(6 \times 6) + (6 \times 6) \\
(6 \times 2) &= (6 \times 6)(6 \times 2)\left[(2 \times 6)(6 \times 6)(6 \times 2) + (2 \times 2)\right]^{-1} \\
(6 \times 1) &= (6 \times 1) + (6 \times 2)\left[(2 \times 1) - (2 \times 6)(6 \times 1)\right] \\
(6 \times 6) &= \left[(6 \times 6) - (6 \times 2)(2 \times 6)\right](6 \times 6)
\end{aligned}$$

5.2.4 Implementation

The above EKF algorithm has been implemented for a straight line trajectory. Because of its slow convergence, it was repeated 100 times. The results of the estimation are shown in Figure 5.3 and a trajectory plot after the estimation is shown in Figure 5.4. The trajectory is shown for encoder data from a "new" trajectory, i.e., one on which the estimator has not

trained. The latter shows the believed trajectory before calibration denoted “Uncalibrated” and the same trajectory using the calibrated parameters denoted “filtered.”

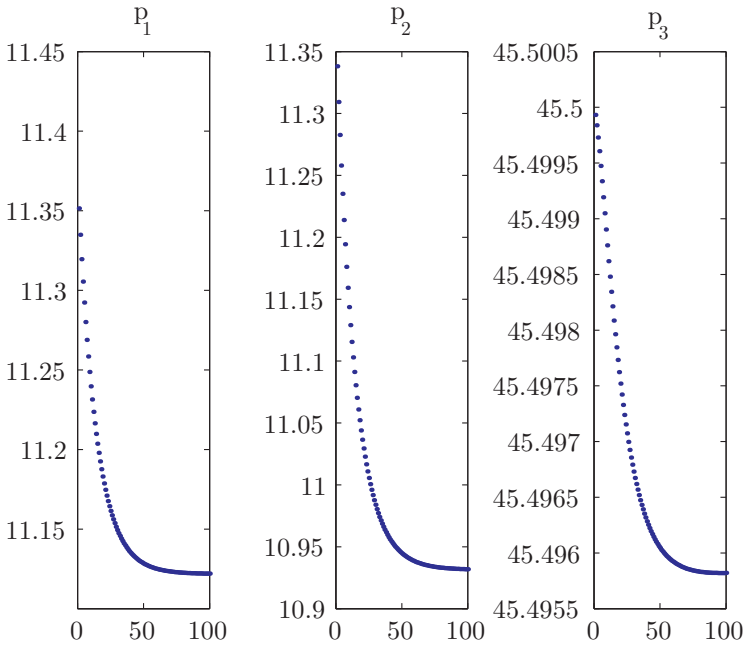


Figure 5.3: Parameters estimated from a straight line trajectory using the EKF. The x-axis is the number of repititions of the EKF. The y-axis is the value of each parameter. The parameters converged.

5.3 Validation

After running the EKF process over the same data (the square trajectories) that the UMBmark test used, the estimated set of parameters is shown in Table 5.2. We compare the two calibration methods by using each set of parameters with the same set of encoder data and measuring the squared error of the final states from the ground truth data. Figure 5.5 shows both sets of parameters pushed through a square trajectory - one of the tests that was used to calibrate - and the squared errors are shown beside it. Of course, the previous test is an exmaple of cross-validation, so we push the parameters through a “new” trajectory - a straight line trajectory. These results are shown in Figure 5.6 with associated errors.

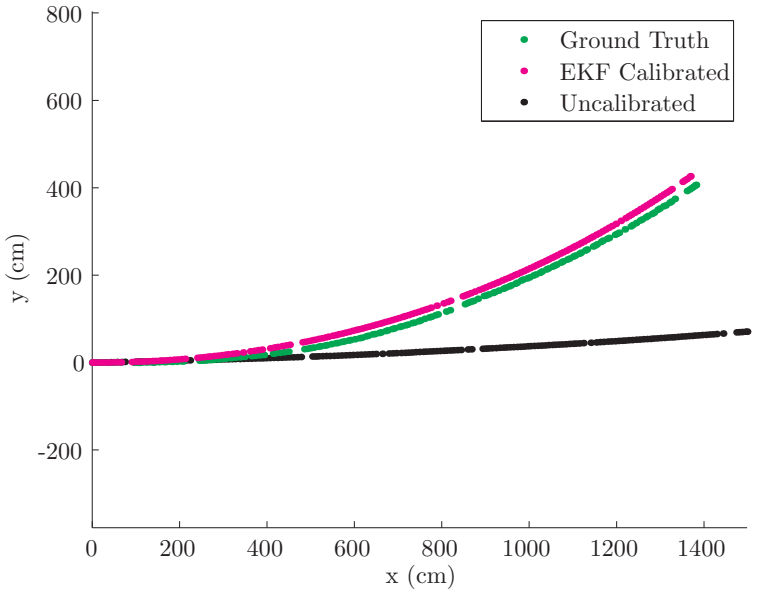
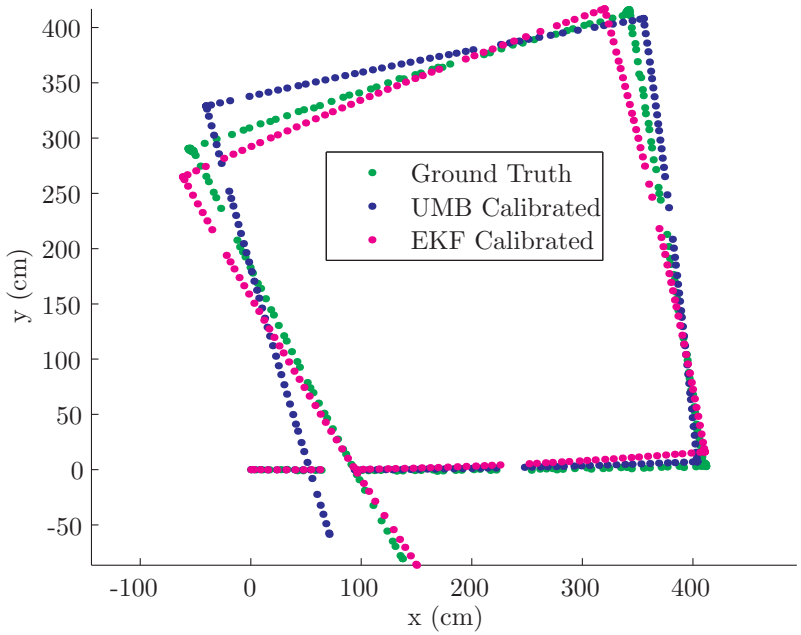


Figure 5.4: A trajectory using the parameters estimated by the EKF. The believed trajectory before calibration is in black and the same trajectory using the calibrated parameters is in red.

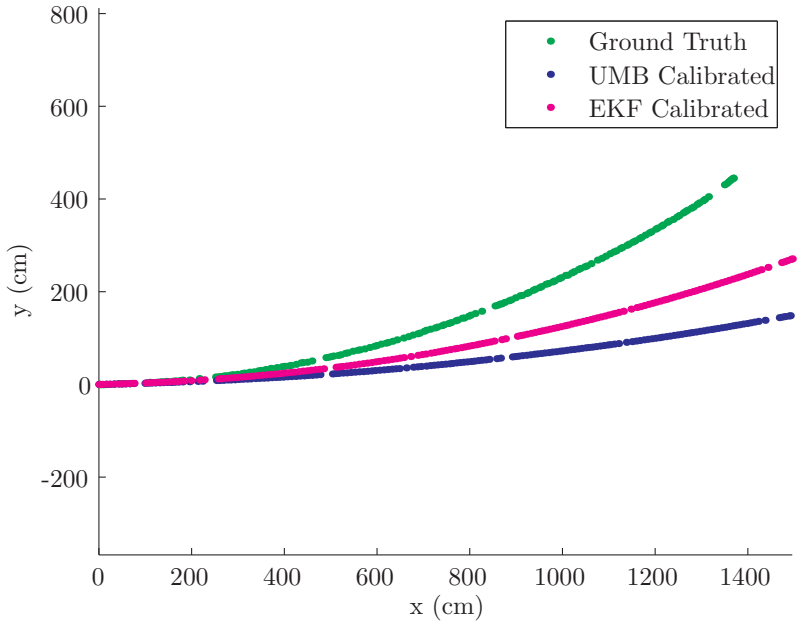
	p_1	p_2	p_3
UMB	11.39	11.35	44.02
EKF	11.61	11.53	43.78

Table 5.2: Calibrated odometric parameters using the UMBmark test and the EKF.



UMB Error = 70 cm
EKF Error = 14 cm

Figure 5.5: Comparing calibration methods by evaluating identical inputs (encoder data) with the parameter set as computed by each calibration method. The "error" values are the Euclidean distances between the predicted final positions and the ground truth final position. The UMBmark parameters yield an error ≈ 5 times that of the EKF. The EKF is a more accurate method of calibration.



UMB Error = 321 cm
EKF Error = 216 cm

Figure 5.6: Comparing calibration methods on a "new" set of inputs (avoiding cross-validation). The "error" values are the Euclidean distances between the predicted final positions and the ground truth final position. The EKF is again closer to the ground truth data.

Chapter 6

Estimation and Control

Closed-loop control policies typically outperform open-loop policies because of the use of feedback from a state estimator. Sensor measurements are fused with a state prediction—a prior distribution over the state—in a filter to compute the state estimate—the posterior distribution. For example, a mobile robot using only encoders to follow a straight line path in a system with uncertainty often accumulates drift because of a poor state estimate. As discussed earlier, this may be due to dead reckoning error and the fact that encoders are locally referenced sensors. If the robot restarts with an added exteroceptive sensor, e.g., GPS, the GPS measurements are fused with the encoder measurements and result in a more accurate state estimate.

Robotic systems generally rely on several levels of feedback control to achieve a task. For example, at the highest level, perhaps a robot controller is deciding which trajectory to follow in order to complete a task. Then, the desired trajectory is an input to a trajectory tracking controller which generates vehicle inputs such as a foward velocity and a turn rate. Then, a lower level controller ensures that the kinematic inputs such as wheel velocities match the vehicle inputs.

In this chapter, we review the linear quadratic Gaussian (LQG) optimal controller (Section 6.1). We also present a heuristic method of control for SL segments and ZPTs (Section B.1), a method of path following (B.2), and a heuristic controller track a set of waypoints (B.3). These controllers are used for different representations of a reference path. Finally, we explain the state estimators and controllers for the robot (Section 6.2) and the virtual robot (Section 6.3).

6.1 LQG

A LQG controller is the combination of a linear quadratic estimator, e.g., a Kalman filter, and a linear quadratic regulator. This controller performs optimally for linear, time-invariant systems perturbed by zero-mean Gaussian noise. We model the noise in our system as zero-

mean Gaussian random variables. To deal with the nonlinearities of our system, we linearize. We detail the Kalman filter and the LQR trajectory tracking controller for the robot.

6.1.1 Kalman Filter

The robot computes a state estimate using a Kalman filter with each new measurement. We use an EKF to fuse the sensor information for the nonlinear system. We described the EKF in Section 5.2.2, but in this policy, the robot does not estimate the odometric parameters.

The EKF equations are identical to (5.22) - (5.26), but the matrix definitions differ slightly.

$$G_k = g'(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) = \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \boldsymbol{x}_{k-1}} = \frac{\partial g(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)}{\partial \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}_{k-1}}$$

$$\boldsymbol{G}_k = \begin{bmatrix} 1 & 0 & -\Delta d_k \sin\phi \\ 0 & 1 & -\Delta d_k \cos\phi \\ 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

where $\phi = \left(\theta_{k-1} + \frac{\Delta\theta_k}{2}\right)$. The (3×3) process noise matrix are the odometry error covariance matrices from (4.23) and (4.24). The robot uses the measurement model in Section 4.3.

6.1.2 LQR Controller

An LQR trajectory following strategy can be used to track the path. The primary advantage offered here is that the cost function can be tuned to fit a system. The quadratic cost function to be minimized

$$J_{k+1}(\boldsymbol{x}) = \min_u \left[\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} + \sum_{\boldsymbol{x}'} J_k\left(\boldsymbol{x}'\right) \right]$$

For a deterministic system, this becomes

$$J_{k+1}(\boldsymbol{x}) = \min_u \left[\boldsymbol{x}^T \boldsymbol{Q} \boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R} \boldsymbol{u} + J_k\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}\right) \right]$$

Assuming that J is also quadratic,

$$J_k(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P}_k \boldsymbol{x}$$

Then, J_1, for example, becomes

$$\begin{aligned} J_1(\boldsymbol{x}) &= \min_u \left[\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} + J_0\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}\right)\right] \\ &= \min_u \left[\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u} + \left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}\right)^T\boldsymbol{P}_0\left(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}\right)\right] \end{aligned}$$

Solving for $\boldsymbol{u}$ in the following

$$\frac{\partial J_1}{\partial \boldsymbol{u}} = 0$$

yields the optimal $\boldsymbol{u}$, which is

$$\boldsymbol{u} = -\left(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{P}_0\boldsymbol{B}\right)^{-1}\boldsymbol{B}^T\boldsymbol{P}_0\boldsymbol{A}\boldsymbol{x}$$

Substituting this expression back into $J_1(\boldsymbol{x})$ and generalizing for an iterative approach yields

$$J_k(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{P}_k\boldsymbol{x}$$

where

$$\boldsymbol{K}_k = -\left(\boldsymbol{R} + \boldsymbol{B}^T\boldsymbol{P}_{k-1}\boldsymbol{B}\right)^{-1}\boldsymbol{B}^T\boldsymbol{P}_{k-1}\boldsymbol{A} \tag{6.2}$$

$$\boldsymbol{P}_k = \boldsymbol{Q} + \boldsymbol{K}_k^T\boldsymbol{R}\boldsymbol{K}_k + \left(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}_k\right)^T\boldsymbol{P}_{k-1}\left(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}_k\right) \tag{6.3}$$

Thus, with the following linearized system matrices

$$\boldsymbol{A}_k = \frac{\partial \boldsymbol{f}_k\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)}{\partial \boldsymbol{x}_k}$$

$$\boldsymbol{B}_k = \frac{\partial \boldsymbol{f}_k\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)}{\partial \boldsymbol{u}_k}$$

the system dynamics can be linearized about the target trajectory

$$\boldsymbol{x}_{k+1} \approx \boldsymbol{f}_k\left(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*\right) + \boldsymbol{A}_k\left(\boldsymbol{x}_k - \boldsymbol{x}_k^*\right) + \boldsymbol{B}_k\left(\boldsymbol{u}_k - \boldsymbol{u}_k^*\right)$$

or

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_{k+1}^* \approx \boldsymbol{A}_k\left(\boldsymbol{x}_k - \boldsymbol{x}_k^*\right) + \boldsymbol{B}_k\left(\boldsymbol{u}_k - \boldsymbol{u}_k^*\right)$$

with the resulting control sequence

$$\boldsymbol{u}_k - \boldsymbol{u}_k^* = \boldsymbol{K}_k\left(\boldsymbol{x}_k - \boldsymbol{x}_k^*\right)$$

Loop

1. Compute $\boldsymbol{A}_k\left(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*\right)$ and $\boldsymbol{B}_k\left(\boldsymbol{x}_k^*, \boldsymbol{u}_k^*\right)$
2. Compute $\boldsymbol{K}_i$ from (6.2) and $\boldsymbol{P}_i$ from (6.3)
3. Determine control $\boldsymbol{u}_i$ from (6.4)
4. Update actual state: $\boldsymbol{x}_{k+1} = \boldsymbol{f}\left(\boldsymbol{x}_k, \boldsymbol{u}_k\right)$

Figure 6.1: LQR trajectory following algorithm

which can be rearranged to yield the vehicle control input

$$\boldsymbol{u}_k = \boldsymbol{u}_k^* + \boldsymbol{K}_k\left(\boldsymbol{x}_k - \boldsymbol{x}_k^*\right) \tag{6.4}$$

Thus, the algorithm for LQR trajectory following is outline in Figure 6.1. The linearized $\boldsymbol{A}_k$ and $\boldsymbol{B}_k$ matrices for a differential drive vehicle are computed using the simplified state transition model are are as follows.

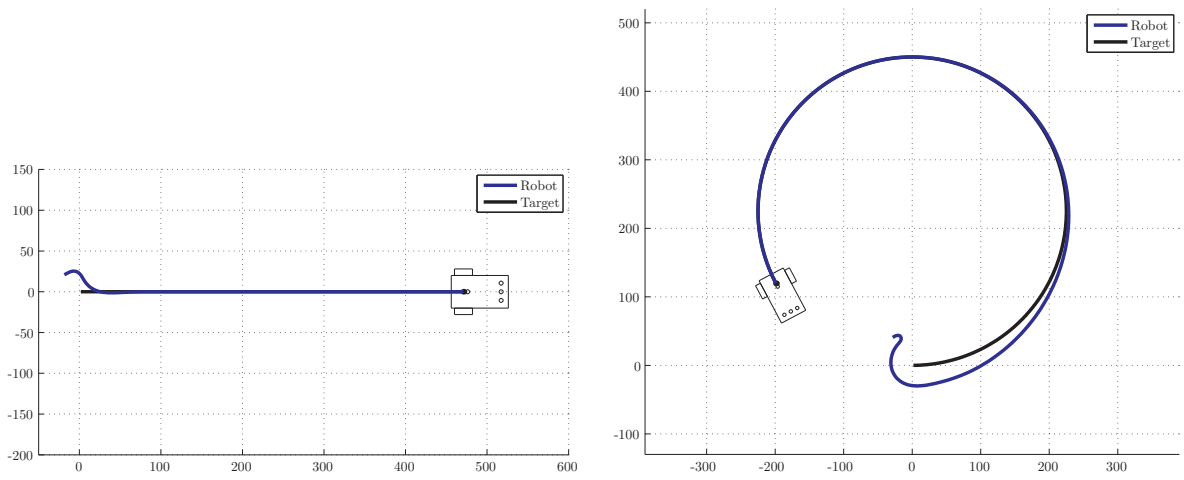
$$\boldsymbol{A}_k = \begin{bmatrix} 1 & 0 & -v_k \Delta t \sin\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) \\ 0 & 1 & v_k \Delta t \cos\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{B}_k = \begin{bmatrix} \Delta t \cos\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) & -\frac{1}{2} v_k \Delta t^2 \sin\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) \\ \Delta t \cos\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) & \frac{1}{2} v_k \Delta t^2 \cos\left(\theta_{k-1} + \frac{\omega_k \Delta t}{2}\right) \\ 0 & \Delta t \end{bmatrix}$$

We implement this algorithm for a differential drive robot following a straight line and a constant curvature arc with offset initial conditions in Figure 6.2.

6.2 Robot

The robot uses the LQG controller as described in Sections 6.1.1 and 6.1.2. We now describe the robot differential drive control.



(a) Straight line.

(b) Constant curvature.

Figure 6.2: LQR trajectory tracking of various trajectories for a differential drive robot with offset initial conditions.

6.2.1 Differential Drive Control

At the lowest level of the control architecture is a coupled proportional-integrator (PI) controller which commands wheel velocities for a differential drive robot given a desired forward velocity and turn rate. A PI controller is wrapped around each wheel so that it maintains its desired velocity. The two wheel controllers are then coupled by another PI controller to maintain the correct heading. This process is diagrammed in Figure 6.3. The algorithm is as follows.

1. read encoders and convert to wheel angles
2. compute individual angular velocities
3. compute feedback forward velocity and turn rate using motion model and odometric parameters
4. evaluate error and integrate
5. compute coupled PI conrol effort and check for saturation

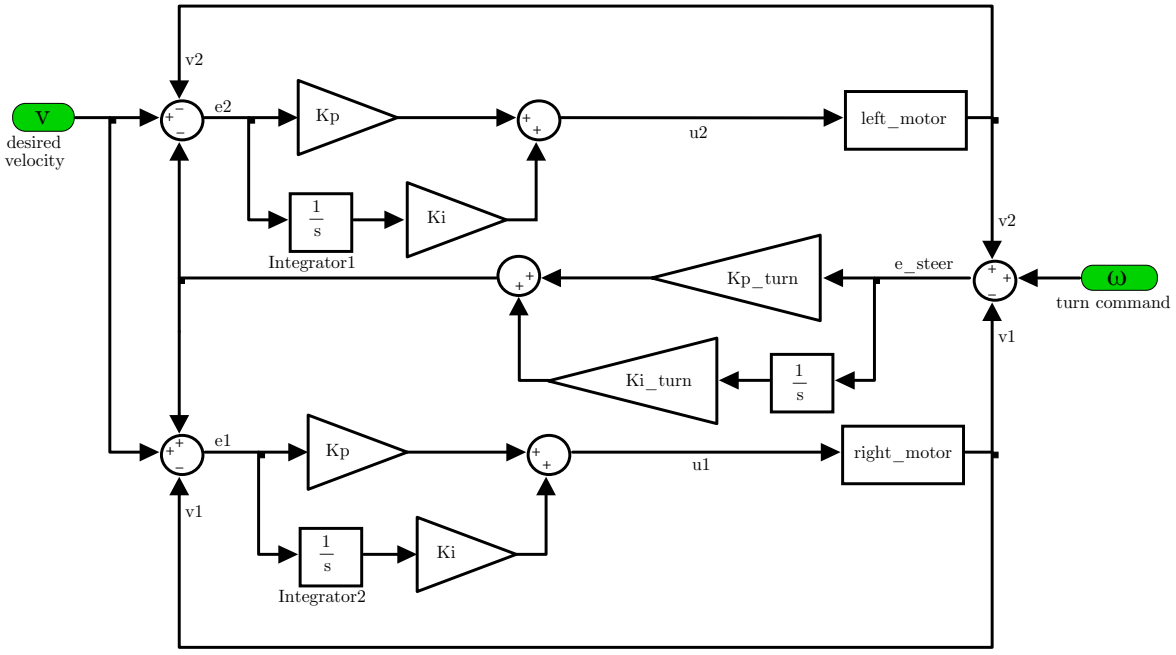


Figure 6.3: Coupled PI controller for a differential drive vehicle. Low-level controller which commands and maintains wheel velocities given a desired robot forward velocity v and a desired turn rate ω. Graphic from [57].

6.3 Virtual Robot

6.3.1 State Estimatation

To track its trajectory, the virtual robot computes a state estimate. We have not implemented a classical approach for virtual robot state estimation, but a heuristic method is as follows. At the start and end of each virtual robot segment, the virtual robot will adopt the robot state estimate. The start position is q_0 and the end position is q_5 in Fig. 4.2. The virtual robot heading is estimated using the method of computing the angle for a circular sector. The circular sector method to compute heading is approximate because the distance traveled by the virtual robot is approximated with a straight line. The radius used here is expressed in (4.9) and is parameterized by (w, ϕ).

Though the circular sector method to compute heading is approximate, the error is sufficiently small, i.e., the virtual robot is able to track a trajectory. While another approach, such as the dot product between the current position, the previous position, and the center of curvature, is more accurate it is outweighed by its computational complexity.

At the start of a new virtual robot primitive,

Loop

1. $x_{vr,0} = x$, $y_{vr,0} = y$
2. Robot follows primitive until believes its position is q_5
   - $x_{vr,1} = x$, $y_{vr,1} = y$
3. $d = \sqrt{(x_{vr,1} - x_{vr,0})^2 + (y_{vr,1} - y_{vr,0})^2}$
4. R is computed from (4.9)
5. $\Delta\theta_{vr,1} = d/R$
6. $\theta_{vr,1} = \theta_{vr,0} + \Delta\theta_{vr,1}$

Figure 6.4: Virtual robot heuristic state estimation algorithm.

6.3.2 Control

One method of The virtual robot uses a LQR trajectory tracking controller as presented in Section 6.1.2. This controller requires a set of references inputs $\{\boldsymbol{u}_0^*, \boldsymbol{u}_1^*, \ldots, \boldsymbol{u}_n^*\}$ and states $\{\boldsymbol{x}_0^*, \boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^*\}$ corresponding to the desired trajectory. Like any state feedback controller, it generates inputs $\boldsymbol{u}_{vr}$ given a current state $\boldsymbol{x}_{vr}$. These inputs must be bounded so that after $\boldsymbol{u}_{vr} \to \boldsymbol{p}_{vr}$, $\boldsymbol{p}_{vr}$ does not exceed $\boldsymbol{p}_{max}$.

The virtual robot can properly follow an arbitrary path with curvature less than its maximum rate of curvature. In Figure 6.5, we demonstrate a straight line trajectory and a constant curvature trajectory for offset initial conditions. In these cases, zero process or measurement noise has been injected. This is to validate the geometric model and trajectory following algorithms, not to validate coverage performance.

6.3.3 Coverage Path Planning: Boustrophedon Decomposition

The global path is planned under the assumption that the virtual robot follows the path with negligible uncertainty. In such systems, the problem of robot coverage is a *coverage path planning* problem. Coverage path planning determines a path such that an agent visits every point in the free space within some fixed distance. A popular approach to coverage path planning is cell decomposition. An *exact cell decomposition* is a representation of the free space which breaks up the free space into non-intersecting regions called cells whose union

fills the free space. Two cells are said to be adjacent if they share a common boundary. An *adjacency graph* encodes cell adjacency where a node corresponds to a cell and an edge connects nodes of adjacent cells. See Figure 6.6.

In this section, we present the boustrophedon coverage planning algorithm [4], which is an exact cell decomposition. The algorithm is stated in Figure 6.7. We make the following assumptions to simplify our analysis.

- bounded, planar workspace and configuration space: $\mathcal{W}, \mathcal{Q} \subset \mathbb{R}^2$
- polygonal boundary and obstacles
- a connected component of free space
- the agent follows the path exactly

We now describe in detail the algorithm and in doing so shall review some definitions.

Step 1. For cell decomposition, we define a *slice* as the preimage of the projection operator π_i, where $\pi_i(q)$ projects a point $q \in \mathcal{Q}$ onto its i^{th} coordinate. That is,

$$\pi_i : \mathcal{Q} \to \mathbb{R}$$

So, we denote a slice as $\mathcal{Q}_\lambda$.

$$\mathcal{Q}_\lambda = \{q \in \mathcal{Q} \mid \pi_1(q) = \lambda\} = \pi_1^{-1}(\lambda)$$

where π_1 is chosen by convention and $\lambda \in \mathbb{R}$. That is, $\mathcal{Q}_\lambda$ is the set of points in the configuration space that have the first coordinate equal to λ, effectively a vertical line at $q_1 = \lambda$ as in Figure 6.8. Varying λ sweeps the slice through $\mathcal{Q}$. So, by collecting slices over all λ, we recover the configuration space. That is,

$$\bigcup_\lambda \mathcal{Q}_\lambda = \mathcal{Q} \,.$$

The portion of the slice in the free space is

$$\mathcal{Q}_{free_\lambda} = \mathcal{Q}_\lambda \cap \mathcal{Q}_{free} \,.$$

As the slice is swept, it intersects or stops intersecting obstacles. We define a *critical point* as a point when the number of connected components of $\mathcal{Q}_{free_\lambda}$ changes. See Figure 6.9.

Slice intervals, $\mathcal{Q}^j_{free_\lambda}$, are connected components of $\mathcal{Q}_{free_\lambda}$. So,

$$\mathcal{Q}_{free_\lambda} = \bigcup_j \mathcal{Q}^j_{free_\lambda}$$

Then, we let I^* denote the set of slice intervals that contain a critical point. Finally, we define a *Morse decomposition* as an exact cell decomposition where cells are connected components of $\mathcal{Q}_{free} \backslash I^*$. Thus, a Morse decomposition has cells that are disjoint and have no holes. So, when I^* is removed, $\mathcal{Q}_{free} \subset \mathbb{R}^2$, but is disconnected.

Step 2. Constructing the adjacency graph occurs concurrently with Step 1. Here we understand how cells are formed and how to incrementally construct the adjacency graph. The slice enters from $q_1 = -\infty$ and an operation occurs at three events: IN, OUT, and MIDDLE. An IN event occurs at a critical point when the number of connected components of $\mathcal{Q}_{free_\lambda}$ increases. A cell is closed and two (or more) new cells are opened. An OUT event occurs at a critical point when the number of connected components of $\mathcal{Q}_{free_\lambda}$ decreases. Two (or more) cells are closed and one is opened. A MIDDLE event occurs at vertices which are not IN or OUT events. Cells are updated by adding a vertex to the list of vertices which define the cell. Cell decomposition and adjacency graph construction are shown in Figure 6.10. An example is shown in Figure 6.6.

Step 3. An exhaustive walk through the adjacency graph can be determined by a depth first search. By exahustive walk, we refer to a path through the graph such that each node is visited at least once. Our implementation of the exhaustive walk and depth first search is given in Algorithms 1 and 2. The resulting walk for the adjacency graph in Figure 6.6 is shown graphically in Figure 6.11 and is the following list of nodes: 1-2-4-2-3-5-6-5-7-5-8-1.

Algorithm 1 ExhaustiveWalk

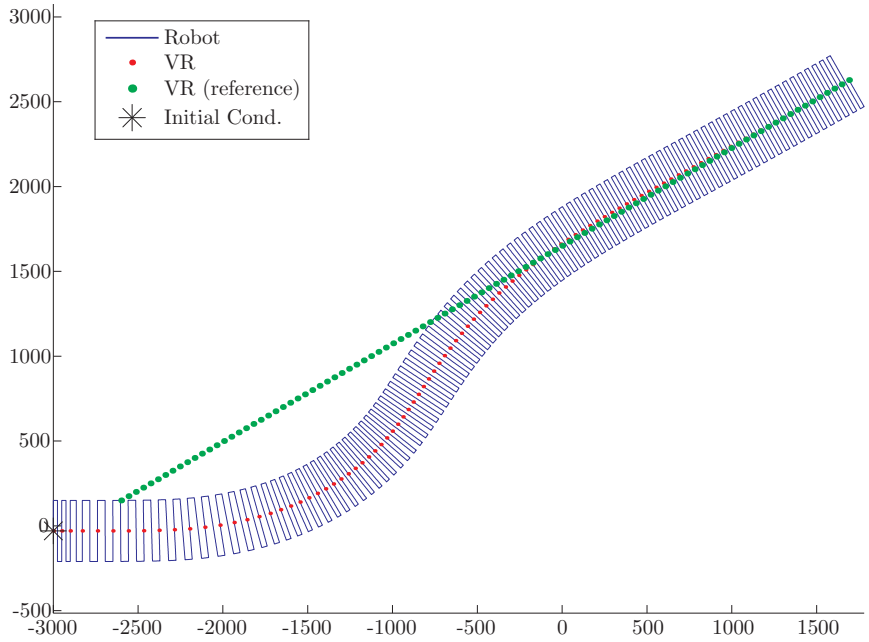
```
1: for i = 1 → |Nodes| do
2:     state[i] ← Unvisited
3: end for
4: path ← runDFS(0, state, {})
```

Step 4. Explicit paths within cells are boustrophedon (back-and-forth) motions. The path follows along a slice until it hits a boundary. Then, it follows the boundary for one width of the coverage implement and passes in the other direction. See Figure 6.11 for an example of an exhaustive walk and a coverage path within a cell.

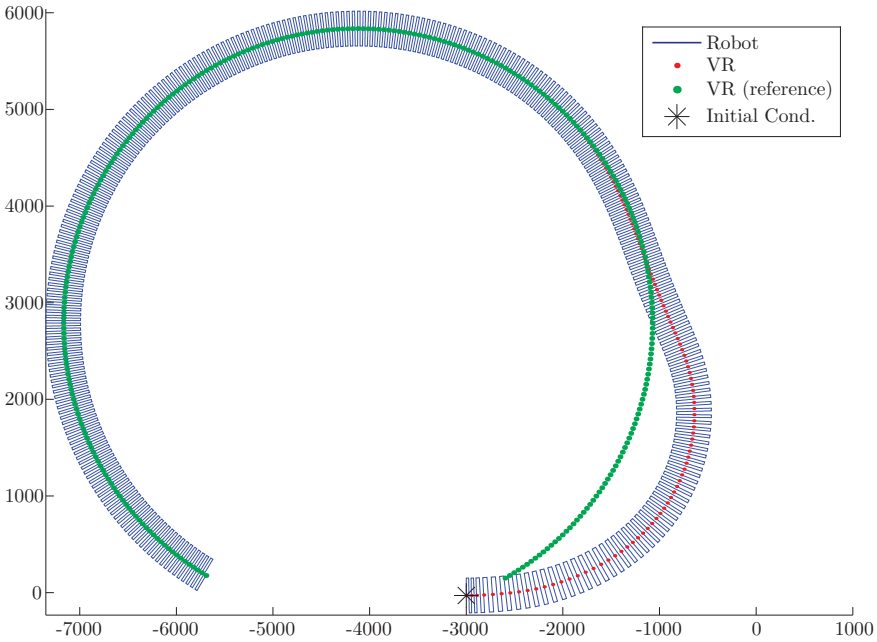
Algorithm 2 RUNDFS(u, state, path)

```
1: state[u] = Visited
2: path ← {path, u}
3: for v = 0 → |Node| do
4:   if isEdge(u, v) & state[v] == Unvisited then
5:     path ← RUNDFS(v, state, path)
6:     if ∃i s.t. state[i]== Unvisited then
7:       path ← {path, u}
8:     end if
9:   end if
10: end for
11: state[u] ← Finished
12: return path
```

Step 5. Connecting cells with motions in $\mathcal{Q}_{free}$ is not explicitly stated. However, it is assumed that a navigation planner plans a path from one cell to another.

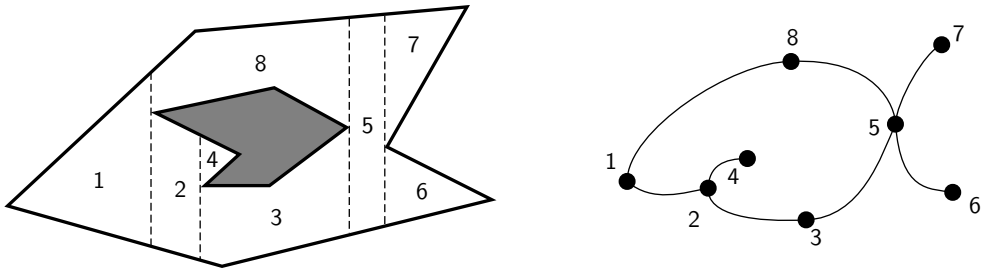


(a) straight line



(b) constant curvature

Figure 6.5: Simulation of the virtual robot tracking a straight line trajectory (top) and a constant curvature trajectory (bottom) from arbitrary initial conditions. Units are arbitrary. No process or measurement noise is injected. This is to validate the geometric model and trajectory tracking algorithm, but is not to validate coverage performance.



(a) exact cell decomposition (b) adjacency graph

Figure 6.6: A boustrophedon decomposition of a planar workspace. The free space is broken up into non-intersecting regions called cells (left), which are numbered. The adjacency graph encodes the cell adjacency where nodes correspond to cells and edges connect nodes of adjacent cells.

1. cell decomposition
2. construct adjacency graph
3. determine an exhaustive walk through the adjacency graph
4. for each cell, compute explicit robot path
5. connect cells in $\mathcal{Q}$

Figure 6.7: Algorithm for the boustrophedon coverage planning algorithm.

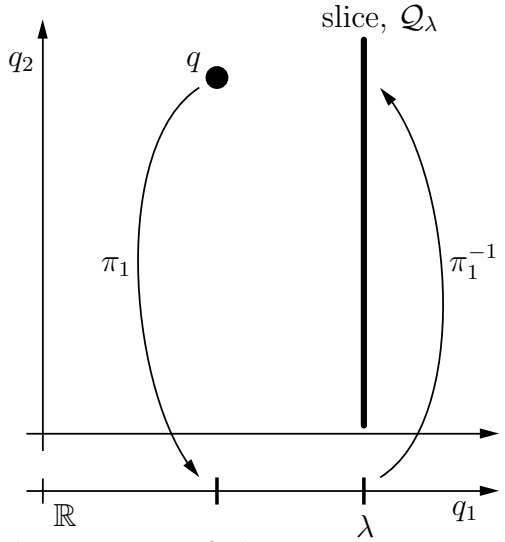
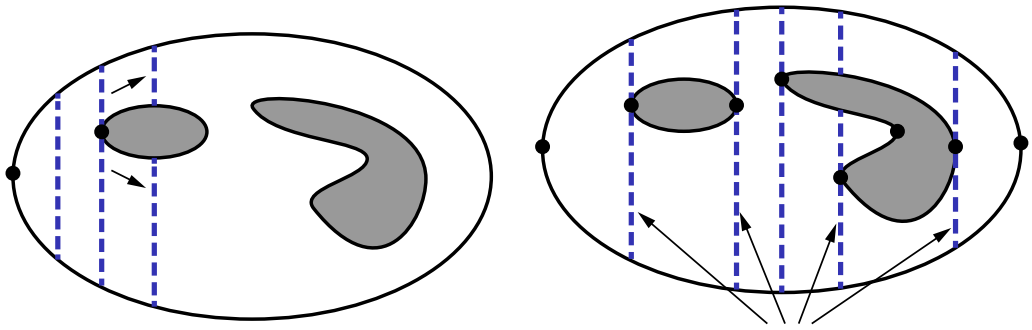
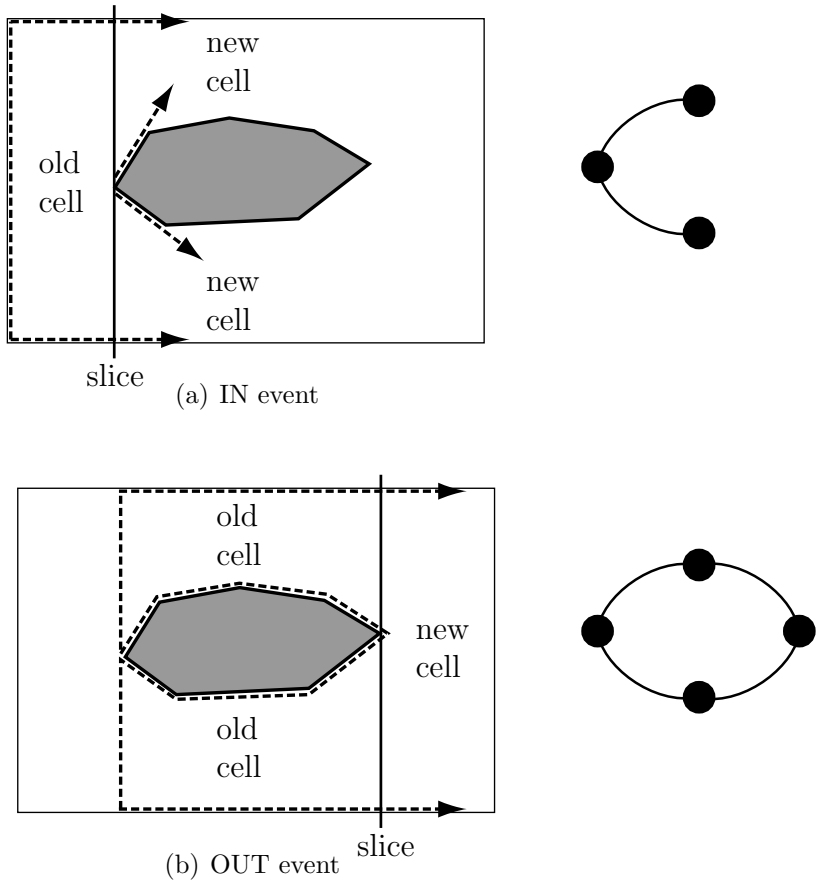


Figure 6.8: A slice, $\mathcal{Q}_\lambda$, is the preimage of the projection operator π_1, which projects a point q onto its first coordinate. The slice is effectively a vertical line. Varying λ moves the slice from left to right.



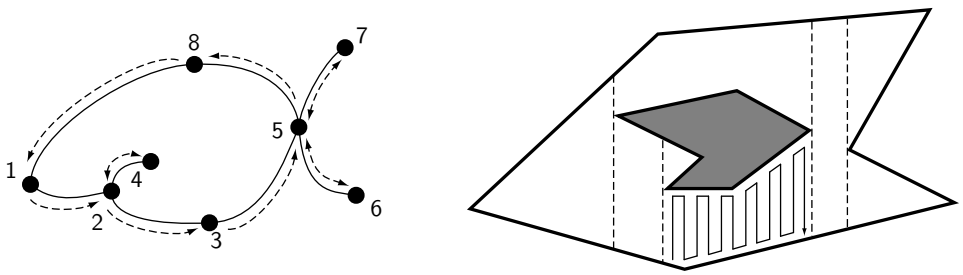
(a) Critical points (b) I^*

Figure 6.9: Critical points (black dots) occur when the number of connected components of $\mathcal{Q}_{free_\lambda}$ (blue dashed lines) changes (left). As the slice enters the free space, it changes from 0- to 1-connected and when it encounters the first obstcale it changes from 1- to 2-connected. I^* is the set of slice intervals that contain a critical point (right).



(a) IN event (b) OUT event

Figure 6.10: Cell decomposition and adjacency graph construction. At an IN event, the old cell is closed and two (or more) new cells are opened. At an OUT event, two (or more) old cells are closed and a new cell is opened (left). When a cell is opened, a node on the adjacency graph is created (right).



(a) Exhaustive walk (b) Exhaustive walk

Figure 6.11: An exhaustive walk through the adjacency graph (left) and an example of the boustrophedon path within a cell.

Chapter 7

Results and Simulation

The virtual robot policy has been implemented in simulation. In Figures 7.2 and 7.1, the virtual robot tracks a constant curvature trajectory and a straight line trajectory for a robot with process and measurement noise.

In order to validate the usefulness of the virtual robot policy, it should be compared to a random strategy and to a systematic approach that uses a LQG controller. Both policies are typical methods of dealing with uncertainty in sensing and actuation, but in this system, LQG control is a different and more classical method of fusing sensor information than the first virtual robot policy. A more standard construction of a path, such as in Section 6.3.3, should be compared to the first virtual robot policy. An LQG controller would be used to follow both paths and thus the performance would indicate which path plan yields better coverage performance.

Perhaps an acceptable method of comparison is the PAC performance measure (Section 2.2.1). This method rigorously tests policy performance, if not in hardware experiment, then in simulation. Unfortunately, we have not implemented these policies on our platform for rigorous simulation. However, we have sample runs of covering a rectangular region for several policies. See Figure 7.3. A LQG controller is a standard method of fusing "locally and globally referenced" sensors and commands the robot to follow a square wave path. The LQG controller uses measurements from the encoders and the GPS sensor. The centerline of the target and actual trajectories match, but the primitives are not parallel and have gaps between them. That is, this controller tracks the global path well, but is locally poor. A "locally referenced" policy uses only encoder information. As expected, there is significant drift globally, but note that the spacing between switchbacks is even. This is surprising because typically more measurements, e.g. GPS, help. The online version of the first "Virtual Robot" policy uses both odometry and GPS information but isolates the locally and globally referenced measurements. The virtual robot yields the highest fraction coverage of the three policies.

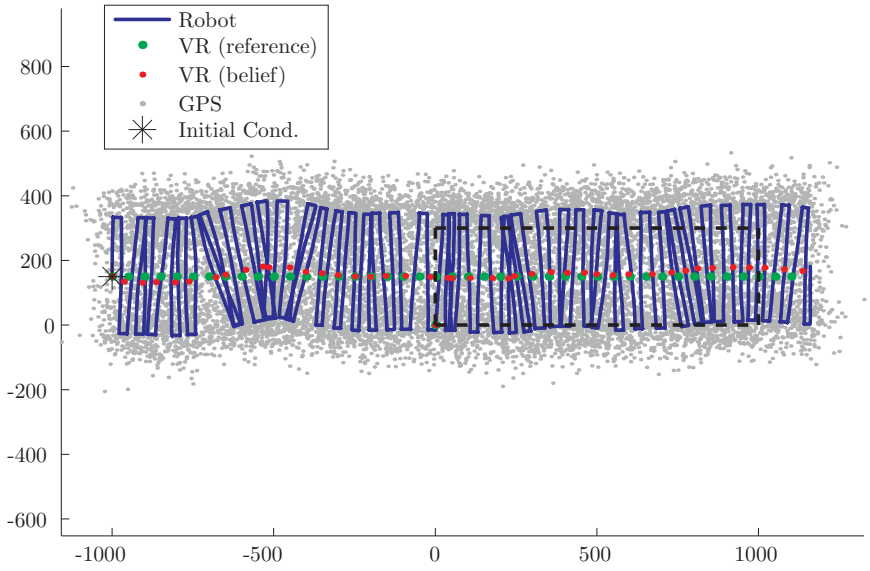


Figure 7.1: Simulation of the virtual robot policy. Units are arbitrary. The virtual robot tracks a straight line trajectory in a system with process and measurement noise such that it attempts to cover the rectangular region (dashed black line).

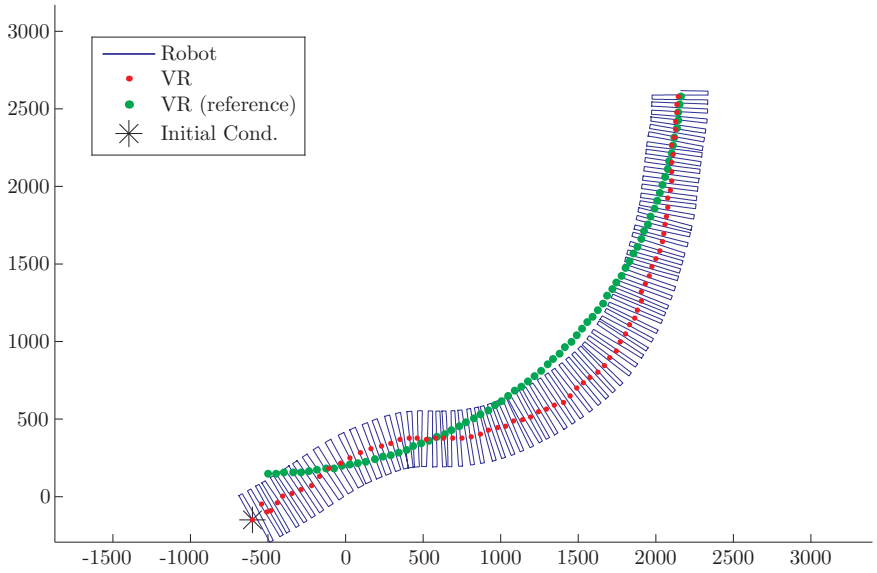
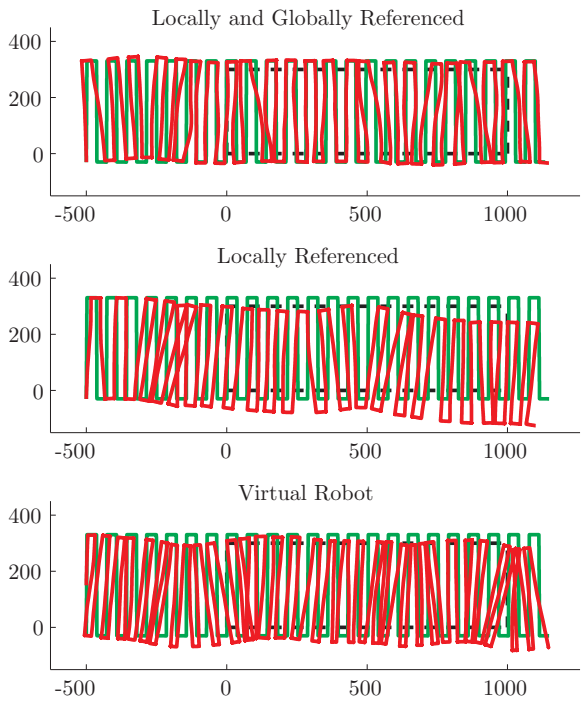
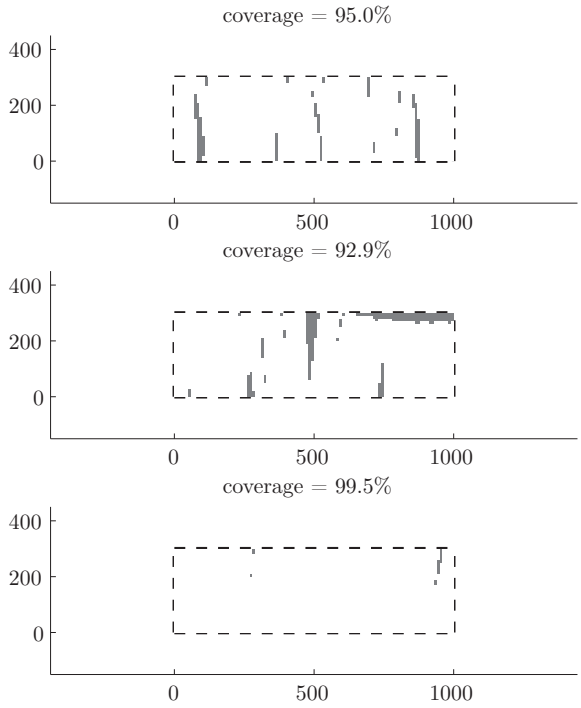


Figure 7.2: Simulation of the virtual robot policy. Units are arbitrary. The virtual robot tracks a constant curvature trajectory for arbitrary initial conditions. The system has injected process and measurement noise.



(a) The reference trajectories are in green and are all identical square wave patterns. The actual trajectories are in red.



(b) Coverage of the desired region after sweeping the coverage implement over the actual (red) path.

Figure 7.3: Simulation of policies using locally and globally referenced sensors. "Locally and Globally Referenced" (top) is a policy that uses encoders and GPS with an LQG controller. Note the good global propogation (it moves from left to right), but poor local coverage (switchback have gaps). "Locally Referenced" is a policy that uses only odometry. Note the good local coverage (even spacing between switchbacks), but the poor global propogation (drift in the y-axis). The "Virtual Robot" policy is the online version of the first virtual robot policy. It isolates locally and globally referenced sensors and yields the highest amount of coverage.

Chapter 8

Extension: Online Calibration

For the particular case of the virtual robot operating along the boundary, we focus on a variant of the coverage problem in which a robot is to cover a portion of the workspace along its boundary. The goal here is to improve performance by taking advantage of the boundary and the robot's boundary sensor. Our solution is for the robot to perform switchback trajectories in which it leaves the boundary, executes a set of inputs, and returns to the boundary. We show that it is possible to estimate the odometric parameters based on encoder measurements and the difference in time between predicted and observed boundary crossings. The accuracy of our estimate will depend on the choice of trajectory (implying an inherent trade-off between the goals of calibration and coverage). We apply a linear analysis of observability to design a sequence of these trajectories that allow both calibration and coverage near the boundary. We validate our approach both in simulation and hardware experiment.

Online calibration is the process of identifying the odometric parameters during operation. This is useful in order to deal with system uncertainties: terrain slope, friction coefficients, aging and wear of the robot, improper calibration, wheel load-up, and wheel sinkage. One way to capture this uncertainy is by assuming drift in the wheel sizes and wheel base of the vehicle. That is, we attempt to group the uncertain system biases into changes in the odometric parameters. This is a form of adaptive control which makes the system more robust to said uncertainties.

8.1 Method

The method of online calibration chosen here is to perform a sequence of switchback trajectories each of which commands the robot to leave and return to a boundary. For simplicity, we assume the boundary is a straight line along the x_1-axis.

The robot begins with known initial conditions $x_2(0) = 0$ and $x_3(0) = 0$, executes a trajectory, and records the encoder measurements $\boldsymbol{s}$ until the boundary sensor detects a

return to the boundary. We would like to estimate $\boldsymbol{p}'$, the true values of $\boldsymbol{p}$, given several of these measurements $\boldsymbol{s}_i, \ldots, \boldsymbol{s}_k$. Our goal is thus to find a $\boldsymbol{p}$ such that we minimize the change in the boundary sensor position y, defined in (4.22) and repeated below for clarity.

$$y = x_2 + a \sin x_3 + b \cos x_3$$

We define the function $f(\boldsymbol{p}, \boldsymbol{s})$ to represent the change in y subject to measurements $\boldsymbol{s}$, parameters $\boldsymbol{p}$, and initial conditions $y(0)$.

$$f(\boldsymbol{p}, \boldsymbol{s}) = y(\boldsymbol{p}, \boldsymbol{s}) - y(0) \tag{8.1}$$

In order to estimate all parameters, we sum the changes in y for each of the k trajectories and we denote this as $h(\boldsymbol{p})$.

$$h(\boldsymbol{p}) = \sum_{i=1}^{k} f(\boldsymbol{p}, \boldsymbol{s}_i)^2$$

Note that the minimum number of trajectories corresponds to the number of parameters we are attempting to estimate, in this case 3. The argument is squared so that we minimize over a convex function.

$$\boldsymbol{p}_{meas} = \arg\min_{\boldsymbol{p}} h(\boldsymbol{p}) \tag{8.2}$$

This process is depicted in Figure 8.1 and 8.2. The algorithm is stated in Figure 3.

A gradient descent algorithm is used to find the parameter set in (8.2). Process noise corrupts the result of the gradient descent algorithm, $\boldsymbol{p}_{meas}$. It is better to treat $\boldsymbol{p}_{meas}$ as a noisy measurement of the true parameter values using either a low pass filter or Kalman filter to combine the measurement with the parameter estimate. Explicitly,

$$\boldsymbol{p}_{j+1} = \boldsymbol{p}_j - \lambda \left(\boldsymbol{p}_j - \boldsymbol{p}_{meas} \right),$$

where $\lambda \in [0, 1]$ is a filter coefficient, $\boldsymbol{p}_j$ is the current believed parameter set, $\boldsymbol{p}_{meas}$ is the estimate from the gradient descent, and $\boldsymbol{p}_{j+1}$ is the updated believed parameter set. A low λ value distrusts the measurement while a large value assumes the parameters vary rapidly.

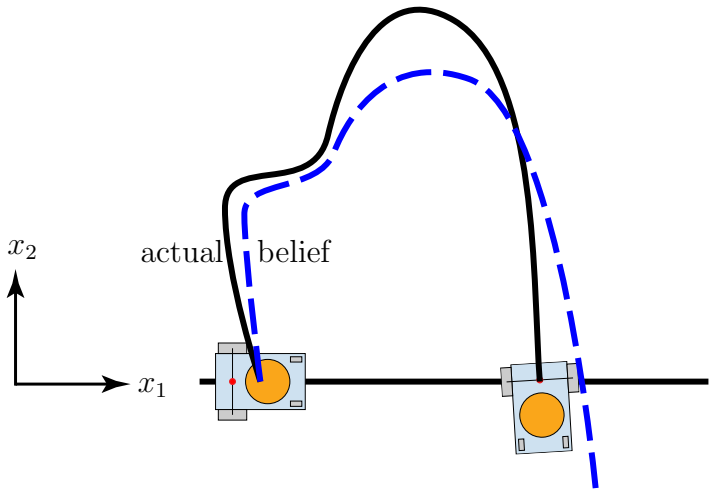


Figure 8.1: An arbitrarily shaped switchback trajectory for online calibration. The boundary sensor (red) is colocated with the center of the wheel axel. The robot stops when the boundary sensor detects a boundary crossing.

Algorithm 3 ONLINE_CALIBRATION($\boldsymbol{p}, \{\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3\}$)

For known initial conditions, $\boldsymbol{x}_0$, a set of believed parameters, $\boldsymbol{p}$, and a set of inputs,$\{\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_3\}$, for three trajectories.

loop
  for $i = 1$ **to** 3 **do**
    EXECUTE_TRAJECTORY($\boldsymbol{u}_i$)
    $\boldsymbol{s}_i \leftarrow$ RECORD_ENCODER_MEASUREMENTS
    if DETECT_BOUNDARY == **true then**
      RESET_POSITION
    end if
  end for
  $h \leftarrow \sum_{i=1}^{k} f(\boldsymbol{p}, \boldsymbol{s}_i)^2$
  $\boldsymbol{p}_{meas} \leftarrow \arg\min_{\boldsymbol{p}} h(\boldsymbol{p})$ // gradient descent
  $\boldsymbol{p} \leftarrow \boldsymbol{p} - \lambda(\boldsymbol{p} - \boldsymbol{p}_{meas})$ // filter measurement
end loop

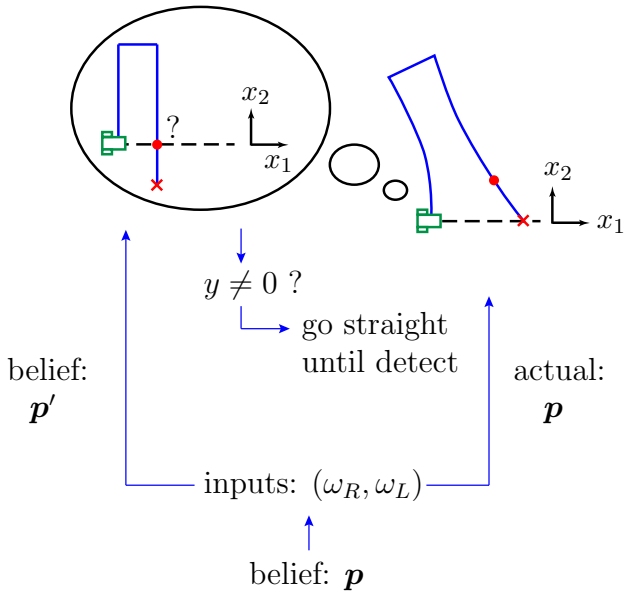


Figure 8.2: Inputs are generated based on believed parameters. Actual trajectory (left) differs from believed trajectory (right). If robot does not detect boundary when it believes it should, it continues to command a straight trajectory. The measurement of $y = 0$ is taken upon boundary detection.

8.2 Observability

We perform observability analyses to check whether a given set of measurements is sufficient to solve for $\boldsymbol{p}$. For discrete measurements, such as during switchbacks, we apply a linear analysis and for a repeated set of measurements, such as distance readings from a laser range finder, we apply Lie Algebra. We assume that the boundary sensor is colocated with the center of the wheel axel until Section 8.2.4.

8.2.1 Discrete Measurements

Consider the system in (4.1). We are given the initial conditions $\boldsymbol{x}(0) = 0$, and so $y(0) = 0$ as well. For a known input $\boldsymbol{u}(t)$, we can measure the first time s at which we again have $y(s) = 0$. We would like to know if it is possible to estimate $\boldsymbol{p}'$ given several of these measurements $s_1, \ldots, s_k$. Let us denote the corresponding input by $\boldsymbol{u}_i(t)$ and output by $y_i(t, \boldsymbol{p})$ for each $i = 1, \ldots, k$, where we have made the dependence of each y_i on $\boldsymbol{p}$ explicit.

Then, a sufficient condition is that the $k \times 3$ Jacobian matrix

$$J(\boldsymbol{p}') = \begin{bmatrix} \partial y_1(s_1, \boldsymbol{p})/\partial \boldsymbol{p} \\ \vdots \\ \partial y_k(s_k, \boldsymbol{p})/\partial \boldsymbol{p} \end{bmatrix}_{\boldsymbol{p}=\boldsymbol{p}'} \tag{8.3}$$

be full rank. If so, then there must exist $\epsilon > 0$ such that

$$y_i(s_i, \boldsymbol{p}) \neq y_i(s_i, \boldsymbol{p}')$$

for all $i = 1, \ldots, k$ and for all $\boldsymbol{p} \in \mathbb{R}^3$ satisfying

$$||\boldsymbol{p} - \boldsymbol{p}'|| < \epsilon$$

and

$$\boldsymbol{p} \neq \boldsymbol{p}'$$

This condition implies that $\boldsymbol{p}'$ is distinguishable. It is equivalent to saying that there exists no nearby parameter vector $\boldsymbol{p} = \boldsymbol{p}' + \delta\boldsymbol{p}$ that would have produced the same measurements $s_1, \ldots, s_k$.

We would like to compute the Jacobian matrix $J(\boldsymbol{p}')$. First, we define

$$z_{ij} = \frac{\partial x_i}{\partial p_j},$$

where it is clear that $z_{ij}(0) = 0$ for all i and j. Notice that

$$\begin{aligned} \dot{z}_{1k} &= \frac{\partial v}{\partial p_k} \cos x_3 - z_{3k} v \sin x_3 \\ \dot{z}_{2k} &= \frac{\partial v}{\partial p_k} \sin x_3 + z_{3k} v \cos x_3 \\ \dot{z}_{3k} &= \frac{\partial w}{\partial p_k} \end{aligned}$$

for each $k = 1, 2, 3$. We may write this as the following matrix differential equation:

$$\dot{Z} = \begin{bmatrix} 0 & 0 & -v \sin x_3 \\ 0 & 0 & v \cos x_3 \\ 0 & 0 & 0 \end{bmatrix} Z + \begin{bmatrix} (u_1/2) \cos x_3 & (u_2/2) \cos x_3 & 0 \\ (u_1/2) \sin x_3 & (u_2/2) \sin x_3 & 0 \\ u_1/p_3 & -u_2/p_3 & -w/p_3 \end{bmatrix}$$

where

$$\Big[Z\Big]_{ij} = z_{ij}$$

and, as stated above, we have $Z(0) = 0$. We may integrate this matrix differential equation together with (4.1) to find $\partial x_i/\partial \boldsymbol{p}_j$ at, for example, the measured time $t = s$. It is then easy to compute

$$\frac{\partial y}{\partial \boldsymbol{p}} = \frac{\partial x_2}{\partial \boldsymbol{p}} + (a\cos x_3)\,\frac{\partial x_3}{\partial \boldsymbol{p}},$$

where $\partial x_2/\partial \boldsymbol{p}$ is the second row of Z and $\partial x_3/\partial \boldsymbol{p}$ is the third row of Z. We do this once to find each row of $J(\boldsymbol{p}')$. It is important to emphasize that, although we can find $J(\boldsymbol{p}')$ to arbitrary precision using this approach, we are certainly not doing "exact" computation. As a consequence, it is not appropriate to apply, for example, MATLAB's `rank` function to check the rank of J. Instead, we will look at the singular values of J. The condition number is the ratio of the largest to smallest singular value in the singular value decomposition of a matrix. A linear system, $Ax = b$, is *well-conditioned* and has a low condition number if small changes in A or b produce small changes in x.

The matrix loses rank as its condition number goes to infinity, or in other words as its smallest singular value vanishes. Note that singular vectors corresponding to vanishing singular values are vectors in the null space of J (or nearly so, in the absence of exact computation). A smaller condition number effectively corresponds to a system with a solution that is more stable in $\mathbb{R}^3$ and that is thus more robust to process noise.

8.2.2 Repeated Measurements

In the case that a robot receives repeated measurements in the form of y, we check that a continuous set of measurements is sufficient to estimate $\boldsymbol{p}$. It must be shown that the parameters $\boldsymbol{p}_0$ are distinguishible in the neighborhood of $\boldsymbol{p}_0$. A point $\boldsymbol{p}_0$ (a set $[p_1^0, p_2^0, p_3^0]$) is distinguishible from $\boldsymbol{p}_1$ (a set $[p_1^1, p_2^1, p_3^1]$) if there exists an input function u_0 such that $z(\boldsymbol{p}_0) \neq z(\boldsymbol{p}_1)$. This is proven by showing that the system is *locally weakly observable* - that there exists a neighborhood of $\boldsymbol{p}_0$ such that every $\boldsymbol{p}$ in that neighborhood other than $\boldsymbol{p}_0$ is distinguishable from $\boldsymbol{p}_0$ [58].

This is verified algebraically by testing the *observability rank condition*. For a nonlinear system

$$\begin{aligned} \dot{x} &= f(\boldsymbol{x}, \boldsymbol{u}) \\ z &= h(\boldsymbol{x}) = \Big[h_1(\boldsymbol{x}) \quad h_2(\boldsymbol{x}) \quad \cdots \quad h_p(\boldsymbol{x})\Big]^T \end{aligned} \tag{8.4}$$

define the column vector ℓ.

$$\ell(x_0, u^*) = \begin{bmatrix} L_f^0(h_1) \\ \vdots \\ L_f^0(h_p) \\ \vdots \\ L_f^{n-1}(h_1) \\ \vdots \\ L_f^{n-1}(h_p) \end{bmatrix}$$

for inputs $\boldsymbol{z} = [h_1, \ldots, h_p]$. Or, for a single measurement, h_1, ℓ is defined as follows.

$$\ell(x_0, u^*) = \begin{bmatrix} L_f^0(h_1) \\ \vdots \\ L_f^{n-1}(h_1) \end{bmatrix}$$

where $L_f^n(h_i)$ is a scalar and is the n^{th} Lie derivative of the measurement function h_i along the state transition function f. Now define the matrix, O, and the gradient operator, d, with respect to the state.

$$O = d\ell(x_0, u^*) = \begin{bmatrix} dL_f^0(h_1) \\ \vdots \\ dL_f^{n-1}(h_1) \end{bmatrix} \tag{8.5}$$

Each $dL_f(h_p)$ is a row vector with n elements. The observability rank criterion is satisfied if $\text{rank}(O) = n$, where $n = \text{rank}(\boldsymbol{x})$. The Lie derivatives for nonlinear systems are the time derivatives of the measurement functions.

$$\begin{aligned} z &= h = L_f^0(h) \\ \dot{z} &= \dot{h} = L_f^1(h) \\ \ddot{z} &= \ddot{h} = L_f^2(h) \\ &\vdots \\ z^{(n-1)} &= h^{(n-1)} = L_f^{(n-1)}(h) \end{aligned}$$

For a control-affine system of the following form

$$\dot{x} = \sum_{i=1}^{m} f_i(x) u_i \tag{8.6}$$

the Lie derivatives are computed as follows.

$$\begin{aligned} L^0(h) &= h \\ L_f^1(h) &= \frac{\partial}{\partial x}\left[L_f^0(h)\right] \cdot f = \frac{\partial h}{\partial x} \cdot f \\ L_f^2(h) &= \frac{\partial}{\partial x}\left[L_f^1(h)\right] \cdot f \end{aligned}$$

8.2.3 Differential Drive

The nonlinear system equations for a differential drive system are

$$\begin{aligned} \dot{y} &= v \sin\theta \\ \dot{\theta} &= \omega \\ \dot{p}_1 &= 0 \\ \dot{p}_2 &= 0 \\ \dot{p}_3 &= 0 \end{aligned} \tag{8.7}$$

The state $\boldsymbol{x} = [y, \theta, p_1, p_2, p_3]^T$, the input $\boldsymbol{u} = [w_R, w_L]$, and the (single) measurement is

$$z = h(\boldsymbol{x}) = y$$

$\dot{x}$ has been excluded in (8.7) as it is known that x is not observable with the measurement. The states $\boldsymbol{p}$ have been augmented to test whether they are observable. This system can be treated as a driftless control-affine system. To match the form of (8.6), the linearized

functions with respect to the input from (8.7) become

$$f_1 = \begin{bmatrix} \frac{p_1}{2}\sin(\theta) \\ \frac{p_1}{p_3} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f_2 = \begin{bmatrix} \frac{p_2}{2}\cos(\theta) \\ -\frac{p_2}{p_3} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Thus, the ℓ is chosen initially[1] as

$$\ell(\boldsymbol{p}, \boldsymbol{u}_0) = \begin{bmatrix} L^0(h) \\ L^1_1(h) \\ L^1_2(h) \\ L^2_{11}(h) \\ L^2_{22}(h) \end{bmatrix} = \begin{bmatrix} y \\ \frac{p_1}{2}\sin(\theta) \\ \frac{p_2}{2}\cos(\theta) \\ \frac{p_1^2}{2p_3}\cos(\theta) \\ \frac{p_2^2}{2p_3}\sin(\theta) \end{bmatrix}$$

and the observability matrix, as computed from (8.5) is

$$O = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{p_1}{2}\cos(\theta) & \frac{1}{2}\sin(\theta) & 0 & 0 \\ 0 & -\frac{p_2}{2}\sin(\theta) & 0 & \frac{1}{2}\cos(\theta) & 0 \\ 0 & -\frac{p_1^2}{2p_3}\sin(\theta) & \frac{p_1}{p_3}\cos(\theta) & 0 & -\frac{p_1^2}{2p_3^2}\cos(\theta) \\ 0 & \frac{p_2^2}{2p_3}\cos(\theta) & 0 & \frac{p_2}{p_3}\sin(\theta) & -\frac{p_2^2}{2p_3^2}\sin(\theta) \end{bmatrix} \tag{8.8}$$

So, seeing if O is full rank by finding its determinant

$$\det(O) = \frac{3p_1^2 p_2^2 \sin(2\theta)}{32p_3^3} \tag{8.9}$$

[1]Should this choice of ℓ not work, higher order Lie derivatives are taken until the resulting column vectors are dependent

which is non-zero, full rank, and thus locally weakly observable if the following conditions are met

- $p_1, p_2 \neq 0$
- $p_3 < \infty$
- $\theta \neq 0, \frac{\pi}{2}$

8.2.4 Offset Position

A similar analysis can be done for measuring a position that is offset from the robot coordinates. The nonlinear system associated with this setup is

$$\begin{aligned} \dot{y} &= v \sin\theta + \ell_1 \omega \cos\theta - \ell_2 \omega \sin\theta \\ \dot{\theta} &= \omega \\ \dot{p}_1 &= 0 \\ \dot{p}_2 &= 0 \\ \dot{p}_3 &= 0 \end{aligned} \tag{8.10}$$

where ℓ_1 is the forward displacement and ℓ_2 is the sideways displacement of the sensor as shown in Figure 8.3. Expanding v and ω in $\dot{y}$ and $\dot{\theta}$ to reveal $\boldsymbol{p}$,

$$\begin{aligned} \dot{y} &= \frac{\omega_R p_1 + \omega_L p_2}{2} \sin\theta + \ell_1 \frac{\omega_R p_1 - \omega_L p_2}{p_3} \cos\theta - \ell_2 \frac{\omega_R p_1 - \omega_L p_2}{p_3} \sin\theta \\ \dot{\theta} &= \frac{\omega_R p_1 - \omega_L p_2}{p_3} \end{aligned} \tag{8.11}$$

The state $\boldsymbol{x} = [y, \theta, p_1, p_2, p_3]^T$, the input $\boldsymbol{u} = [w_R, w_L]$, and the (single) measurement is

$$z = h(\boldsymbol{x}) = y$$

This system can be treated as a driftless control-affine system. To match the form of (8.6), the linearized functions with respect to the input become

$$f_1 = \begin{bmatrix} \frac{p_1}{2}\sin(\theta) + \ell_1 \frac{p_1}{p_3}\cos(\theta) - \ell_2 \frac{p_1}{p_3}\sin(\theta) \\ \frac{p_1}{p_3} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$f_2 = \begin{bmatrix} \frac{p_2}{2}\cos(\theta) - \ell_1 \frac{p_2}{p_3}\cos(\theta) + \ell_2 \frac{p_2}{p_3}\sin(\theta) \\ -\frac{p_2}{p_3} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

the result of the determinant of the observability matrix is

$$\begin{aligned} \frac{1}{32p_3^6} p_1^2 p_2^2 \big(4\big(19\ell_2\left(\ell_1^2+\ell_2^2\right) - 12\ell_2^3\cos[2\theta] + \ell_2\left(-3\ell_1^2+\ell_2^2\right)\cos[4\theta] \\ -6\ell_1\left(\ell_1^2+3\ell_2^2\right)\sin[2\theta] - \ell_1\left(\ell_1^2-3\ell_2^2\right)\sin[4\theta]\big) + (15\ell_2 + \ell_2\cos[4\theta] \\ +6\ell_1\sin[2\theta] + \ell_1\sin[4\theta])p_3^2\big) \end{aligned} \quad (8.12)$$

8.3 Ambiguity

The algorithm fails and does not satisfy observability criteria under certain conditions. We show this using two approaches and then use the results to understand one cause of the ambiguity. Given a set of inputs as a function of time $\boldsymbol{s} = (w_R(t), w_L(t))$, (8.1) becomes a function only of $\boldsymbol{p}$. Assume there exists some $\boldsymbol{p}$ such that $f(\boldsymbol{p}) = 0$. We want to know if there exists $\boldsymbol{p}' \neq \boldsymbol{p}$ such that $f(\boldsymbol{p}') = 0$ as well. In fact, it is easy to show that $f(k\boldsymbol{p}) = 0$ for any $k > 0$. We will do this by showing that

$$\nabla f \cdot \boldsymbol{p} = 0$$

for any $\boldsymbol{p}$ such that $f(\boldsymbol{p}) = 0$, or in other words that the directional derivative of f along $\boldsymbol{p}$ is zero.

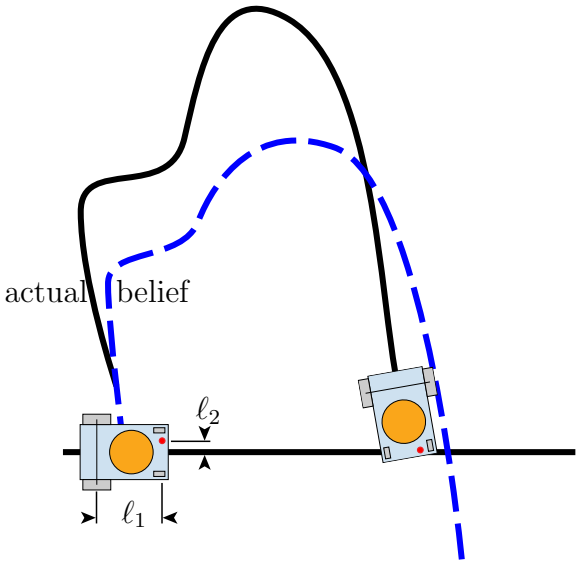


Figure 8.3: Switchback trajectory for a sensor with an offset position. The sensor is displaced forward ℓ_1 on the robot and sideways ℓ_2.

8.3.1 Position Tracking: Approach 1

We begin by noting that

$$f(\boldsymbol{p}) = \int_0^T v \sin\theta \, \mathrm{dt}$$

and so

$$\begin{aligned}
\frac{\partial f}{\partial p_i} &= \frac{\partial}{\partial p_i} \int_0^T v \sin\theta \, \mathrm{dt} \\
&= \int_0^T \frac{\partial \left(v \sin\theta \right)}{\partial p_i} \, \mathrm{dt} \\
&= \int_0^T \left(\frac{\partial v}{\partial p_i} \sin\theta + \frac{\partial \theta}{\partial p_i} v \cos\theta \right) \, \mathrm{dt}
\end{aligned}$$

for each $i = 1, 2, 3$. As a consequence,

$$\nabla f \cdot \boldsymbol{p} = \int_0^T \left((\nabla v \cdot \boldsymbol{p}) \sin\theta + (\nabla\theta \cdot \boldsymbol{p}) \, v \cos\theta \right) \, \mathrm{dt} \, .$$

If we can show that

$$\nabla v \cdot \boldsymbol{p} = cv \tag{8.13}$$

for any $c \neq 0$ and that

$$\nabla\theta \cdot \boldsymbol{p} = 0, \tag{8.14}$$

then we can immediately conclude that

$$\begin{aligned}\nabla f \cdot \boldsymbol{p} &= \int_0^T \left((cv)\sin\theta + (0)v\cos\theta\right)\,\mathrm{dt} \\ &= c\int_0^T v\sin\theta\,\mathrm{dt} \\ &= cf(\boldsymbol{p}) \\ &= 0,\end{aligned}$$

as desired. First, we have

$$\begin{aligned}\nabla v \cdot \boldsymbol{p} &= \begin{bmatrix} w_R/2 \\ w_L/2 \\ 0 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \\ &= \frac{w_R p_1}{2} + \frac{w_L p_2}{2} \\ &= v,\end{aligned}$$

verifying (8.13). Second, to verify (8.14), we begin by noting that

$$\theta(t) - \theta(0) = \int_0^t w\,\mathrm{dt}\,,$$

and so

$$\begin{aligned}\frac{\partial\theta}{\partial p_i} &= \frac{\partial}{\partial p_i}\left(\theta(0) + \int_0^t w\,\mathrm{dt}\right) \\ &= \int_0^t \frac{\partial w}{\partial p_i}\,\mathrm{dt}\end{aligned}$$

for each $i = 1, 2, 3$. As a consequence,

$$\nabla\theta \cdot \boldsymbol{p} = \int_0^t \left(\nabla w \cdot \boldsymbol{p}\right)\,\mathrm{dt}\,.$$

We compute

$$\begin{aligned}
\nabla w \cdot \boldsymbol{p} &= \begin{bmatrix} w_R/p_3 \\ -w_L/p_3 \\ -\left(w_R p_1 - w_L p_2\right)/p_3^2 \end{bmatrix}^T \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} \\
&= \frac{w_R p_1}{p_3} - \frac{w_L p_2}{p_3} - \left(\frac{w_R p_1 - w_L p_2}{p_3^2}\right) p_3 \\
&= \left(\frac{w_R p_1 - w_L p_2}{p_3}\right) - \left(\frac{w_R p_1 - w_L p_2}{p_3}\right) \\
&= 0,
\end{aligned}$$

so clearly

$$\nabla \theta \cdot \boldsymbol{p} = 0$$

also, and we have our result. An example of poor conditioning arises when the sensor is collocated with the center of the wheelbase ($\ell_1, \ell_2 = 0$). This implies that the Jacobian is rank deficient and thus $\boldsymbol{p}'$ is one of many solutions.

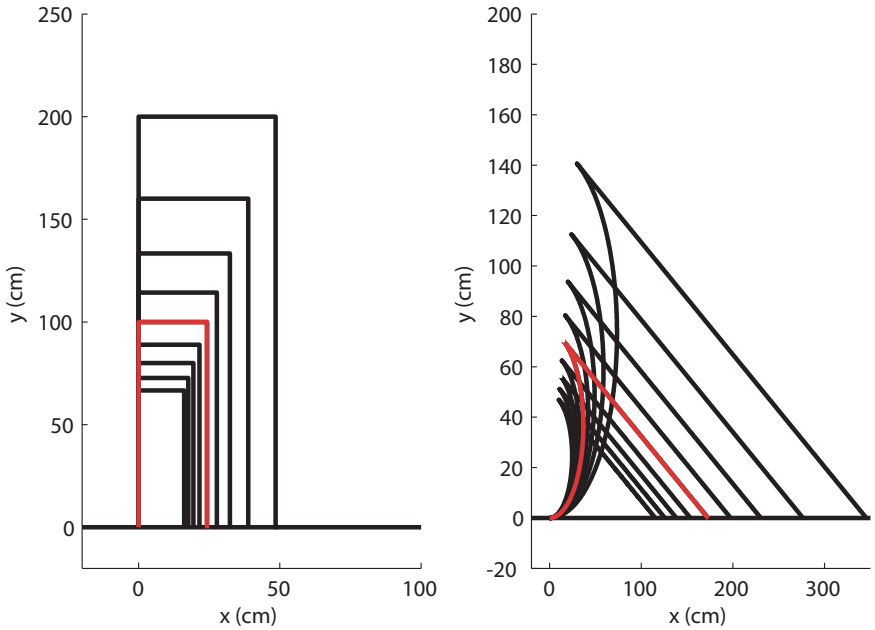


Figure 8.4: When the sensor is collocated with the center of the wheelbase ($\ell_1, \ell_2 = 0$), the true parameters are not distinguishable. Though the actual path is shown in red, there are an infinite number of other parameter values that return the same readings (several are shown in black).

8.3.2 Position Tracking: Approach 2

Again, we note that

$$f(\boldsymbol{p}) = \int_0^T v \sin\theta \, \mathrm{dt}$$

Notice that if $\boldsymbol{p}' = c\boldsymbol{p}$ for some $c > 0$, that is, the parameters are scaled by some constant, then from (4.2)

$$v' = cv$$
$$w' = w$$

So, evaluating $f(\boldsymbol{p}, u)$ at with $\boldsymbol{p}'$,

$$f(c\boldsymbol{p}, u) = \int_0^T v' \sin\theta' \, \mathrm{dt}$$

but

$$\begin{aligned}\theta' &= \theta'(0) + \int_0^T w' \, \mathrm{dt} \\ &= \theta(0) + \int_0^T w \, \mathrm{dt} \\ &= \theta\end{aligned}$$

As a consequence,

$$\begin{aligned}f(c\boldsymbol{p}, u) &= \int_0^T v' \sin\left(\theta(0) + \int_0^t w' \, \mathrm{dt}\right) \mathrm{dt} \\ &= \int_0^T cv \sin\left(\theta(0) + \int_0^t w \, \mathrm{dt}\right) \mathrm{dt} \\ &= c\int_0^T v \sin\theta \, \mathrm{dt} \\ &= cf(\boldsymbol{p}, u)\end{aligned}$$

Thus, if we know that $y(T) = y(0) = 0$, then any $c > 0$ will satisfy $f(c\boldsymbol{p}, u)$ for any u that brings the robot away from and to the boundary. Thus, any switchback trajectory that returns the robot y-position to $y = 0$ will inherently have a scalar ambiguity in the parameters that satisfy its inputs.

8.3.3 Offset Tracking

If instead of tracking or trying to move the robot y-position to and from the boundary, this could be performed for some other point or sensor on the vehicle. This is depicted in Figure 8.3. Consider some point on the robot, offset from the robot (x, y) coordinate, whose y-position, y_s, would be described by

$$y_s = y + \ell_1 \sin(\theta) + \ell_2 \cos(\theta)$$

where ℓ_1 and ℓ_2 are constant, known, dimensions of the robot. Then,

$$\begin{aligned} f(\boldsymbol{p}, u) &= \int_0^T \big(v \sin(\theta)\big)\, \mathrm{dt} \; + \ell_1 \sin(\theta) + \ell_2 \cos(\theta) \\ &= \underbrace{\int_0^T \big(v \sin(\theta)\big)\, \mathrm{dt}}_{f_1(\boldsymbol{p},u)} + \underbrace{\ell_1 \sin(\theta) + \ell_2 \cos(\theta)}_{f_2(\boldsymbol{p},u)} \end{aligned}$$

and using a similar approach as in Section 8.3.2 to test the ambiguity for a scalar set of parameters,

$$f(c\boldsymbol{p}, u) = \int_0^T \big(v' \sin(\theta')\big)\, \mathrm{dt} \; + \ell_1 \sin(\theta') + \ell_2 \cos(\theta')$$

So

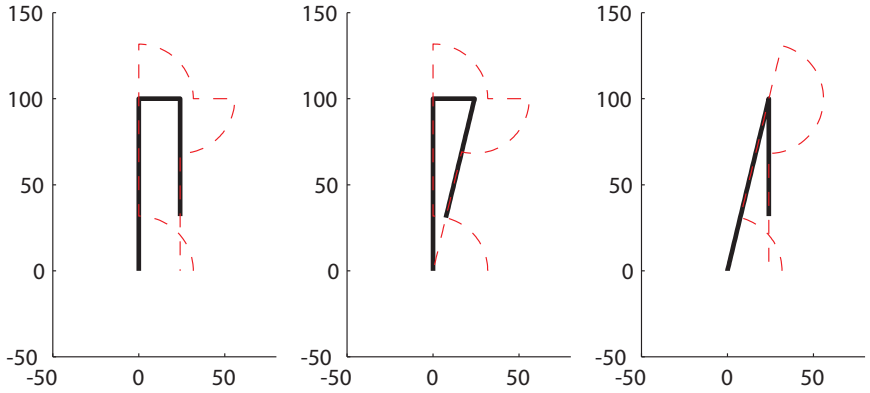
$$\begin{aligned} f(c\boldsymbol{p}, u) &= \int_0^T \big(cv \sin(\theta)\big)\, \mathrm{dt} \; + \ell_1 \sin(\theta) + \ell_2 \cos(\theta) \\ &= c f_1(\boldsymbol{p}, u) + f_2(\boldsymbol{p}, u) \end{aligned}$$

Thus, by tracking some fixed point on the vehicle other than the robot position, there exists *no* scalar ambiguity when the robot leaves and returns to the boundary. This is valid for any point such that $\ell_2 \neq 0$ so that $y_s(0) = \ell_2 = \text{constant}$.

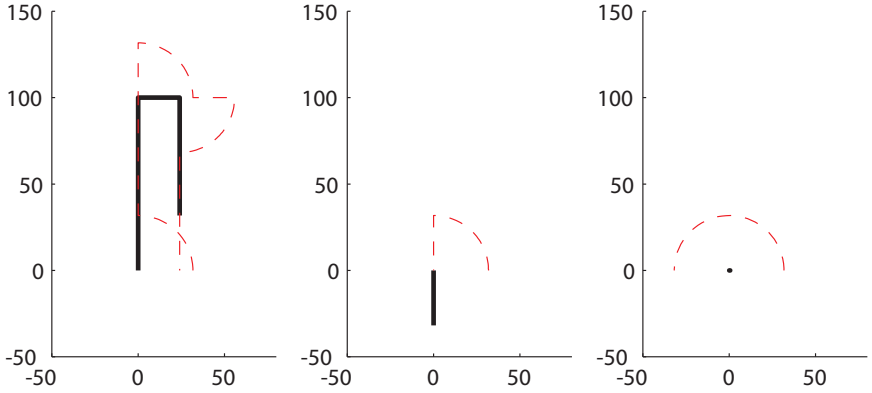
To make it clear, for some $\ell_1, \ell_2 \neq 0$,

$$\begin{aligned} f(c\boldsymbol{p}, u) &= c f_1(\boldsymbol{p}, u) + f_2(\boldsymbol{p}, u) \\ f_1(\boldsymbol{p}, u) + f_2(\boldsymbol{p}, u) &= c f_1(\boldsymbol{p}, u) + f_2(\boldsymbol{p}, u) \\ f_1(\boldsymbol{p}, u) &= c f_1(\boldsymbol{p}, u) \end{aligned}$$

So the robot position is recovered. However, as long as $f_1(\boldsymbol{p}, u) \neq 0$, which is accomplished by trajectory design, $c = 1$ and the scalar ambiguity is no longer an issue. Designing a trajectory such that $f_1(\boldsymbol{p}, u) \neq 0$ requires that the final position of the robot not have both



(a) S_1 (coverage-oriented)



(b) S_2 (calibration-oriented)

Figure 8.5: Three maneuvers for two trajectory sequences. Each maneuver has initial conditions $y = 0$, $\theta = 0$. The black line is the robot path and the red line is the boundary sensor path. Here, $a > 0$ and $b = 0$. The length and width of the maneuvers are optimized in Section 8.4

the sensor and the robot position on the boundary.

8.4 Trajectories

The Jacobian analysis gives us a tool for designing trajectories. Let the initial estimate of believed parameters be $\boldsymbol{p}_0$. Refining this estimate for the three parameters $\boldsymbol{p}$ requires three separate measurements, $y_i(s_i, \boldsymbol{p}_0)$ for each $i = 1, \ldots, 3$, achieved by performing three independent trajectories.

Using (8.3), trajectories can be varied to find a sequence that forms a well-conditioned system with respect to $\boldsymbol{p}$. Finding the globally optimal trajectory sequence is not in the scope of this work, but we find two local minima. Again, the two goals of coverage and

calibration are in competition. A naive trajectory sequence that prioritizes coverage S_1 is shown in Figure 8.5(a) and a sequence that prioritizes calibration, designed to make (8.3) well-conditioned, S_2 is shown in Figure 8.5(b). The former covers more new territory than the latter, but has a higher condition number. All trajectories begin with $y = 0$, $\theta = 0$ and end when the sensor detects the boundary.

S_1 consists of a rectangular path, a quasi-rectangular paths, and a triangular path, chosen so that the combined movement covers a rectangular patch. S_2 begins with the same rectangular path as S_1. The second trajectory of S_2 is a 90° zero point turn followed by a reverse. The third trajectory of S_2 is simply a zero point turn.

8.4.1 Optimization

The condition numbers for S_1 and S_2 are plotted in Figures 8.6(a) and 8.6(b). The condition number is computed for varying values of the trajectory length, L, and width, W, for S_1 and ${S_2}^2$.

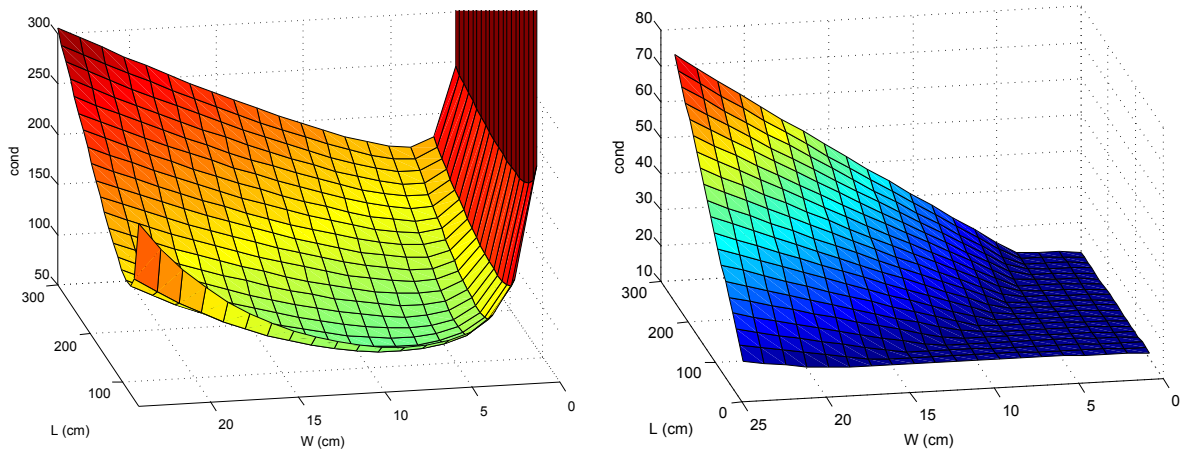
Additionally, L has an upper bound dependent on the parameter uncertainty. This is necessary because drift accumulates with distance traveled. Large errors can cause the robot to never return to the boundary.

8.5 Performance

The performance of the algorithm is rigorously tested in simulation.

8.5.1 Initial Conditions

We compare the effects of different $\boldsymbol{p}_{actual}$ when the robot is started with the same $\boldsymbol{p}_{belief}$ by observing the root-mean squared (RMS) value of the distance error between the actual trajectory and the nominal (believed) trajectory after 1 and 5 iterations. This is plotted in Figure 8.7.



(a) S_1 (coverage-oriented) (b) S_2 (calibration-oriented)

Figure 8.6: Condition numbers for systems of trajectories with varying length, L and width, W applied to the dimensions of the UIUC Segbot. A minimum is clear in the coverage-oriented sequence whereas a minima range of L and W exist, suitable for calibration-oriented sequences. This provides a degree of freedom that can be optimized for coverage tasks.

8.5.2 Process Noise

The effects of increasing process noise on RMS error are compared for a consistent value of initial $\boldsymbol{p}_{belief}$. From Figure 8.8, a robot's likelihood of failure, defined as when the robot does not return to the boundary, increases both with process noise and with iteration.

8.5.3 Boundary Shape

To this point, the results have used straight boundary lines. Of obvious concern is the effect of an arbitrary boundary shape. This section extends our analysis to piecewise, constant curvature boundaries. Process noise was injected as follows

$$\delta\theta_{L/R} = \delta\theta^{e}_{L/R} + \nu$$
$$\nu \sim \mathcal{N}(0, K).$$

This is a modified form of (4.4) where we inject each wheel turn with noise drawn from a normal distribution with constant variance, $K = 0.002$. In Figure 8.9 we apply online calibration to environments with concave and convex boundaries. The curvature induces a

[2]Trajectories two and three of S_2 are not parameterized by L and W.

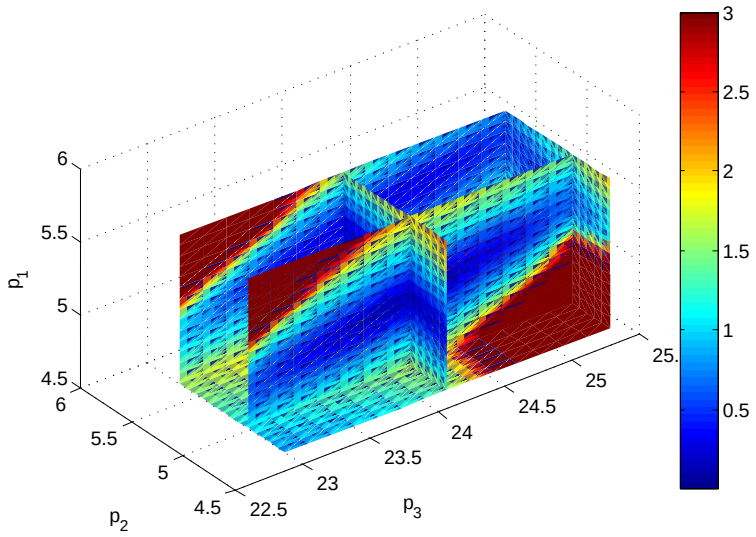


Figure 8.7: RMS error between the nominal and actual trajectories for varying values of $\boldsymbol{p}_{actual}$ after one iteration of online calibration. $\boldsymbol{p}_{belief}$ is initially the same for each trial. Coverage performance degrades as $\boldsymbol{p}_{actual}$ and $\boldsymbol{p}_{belief}$ diverge.

parameter bias and does not break the algorithm. For each case, the algorithm converged and an effective set of parameters was found. While the notion of effective parameters is slightly different here, coverage is maintained - robust to process noise. This is a sufficiently rich demonstration for approximating any continuous smooth boundary.

8.5.4 Time-Varying Parameters

The effective systematic parameters are often dynamic, e.g., wheels deteriorating or loading up with mud, terrain variations, etc. We model this phenomenon by injecting a sinusoidal disturbance to the actual parameters. In Figure 8.10, the convergence properties depend on process noise and initial parameter estimates, but is capable of tracking parameter variations of amplitude $\pm 10\%$ up to frequencies of 1 cycle per 15 iterations.

8.6 Hardware Experiments

8.6.1 Experimental Setup

We applied our motion planning algorithm to a differential-drive robot, the UIUC Segbot, in Figure 8.11, to validate the online calibration algorithm. A reflective sensor is mounted

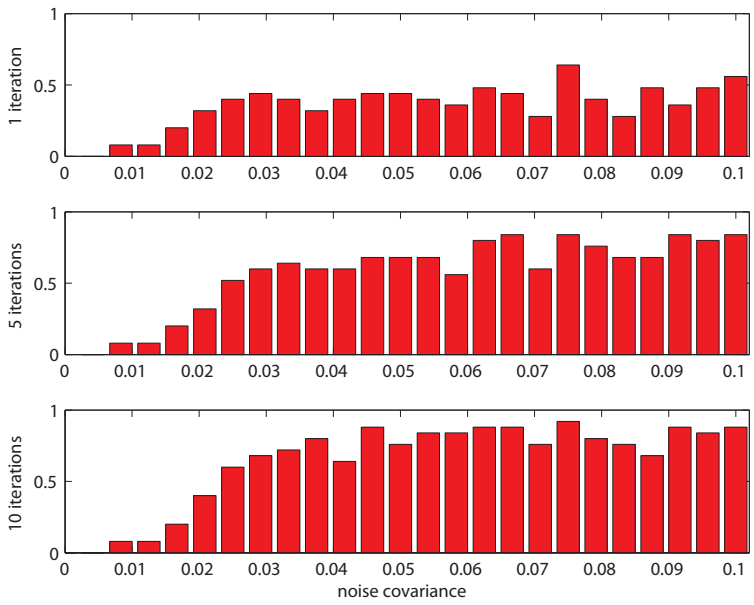


Figure 8.8: A Monte-Carlo analysis of algorithm robustness with respect to process noise. The red bars indicate the probability of failure, defined as when the robot does *not* return to the boundary. The simulation indicates that the likelihood of failure increases both with process noise and with iteration.

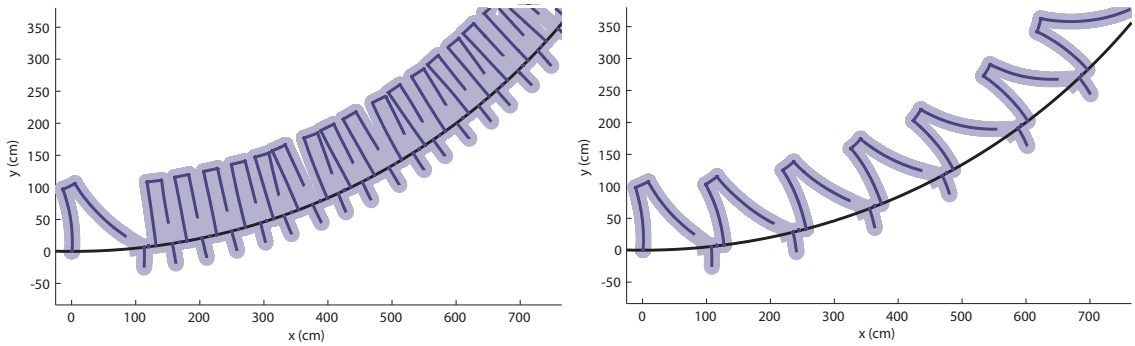
a known distance $a = 32$cm in front of the center of the wheels. A $5 \times 0.3m$ section of IR absorbing material was laid on the floor to serve as the boundary.

The trajectory sequence S_2 in figure 8.5(b) with dimensions $L = 100$cm and $W = 12$cm was chosen from the set that minimizes the condition number of $J(\boldsymbol{p}')$ shown in figure 8.6(b). The Segbot was commanded to line-follow to reset its position to $y = 0$, $\theta = 0$. Then, it backs up to maintain coverage.

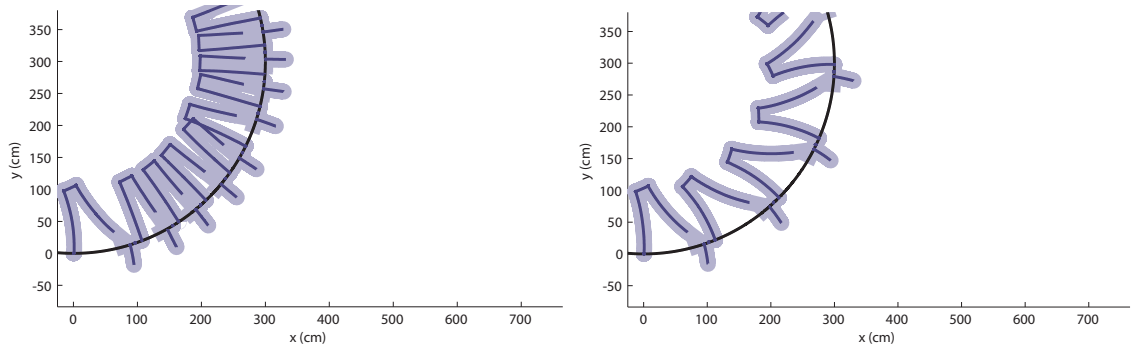
Our hardware experiments consist of initializing the robot to believed p values consisting of every permutation of $[95\%, 100\%, 105\%]$ of $\boldsymbol{p}_{act}$

8.6.2 Results

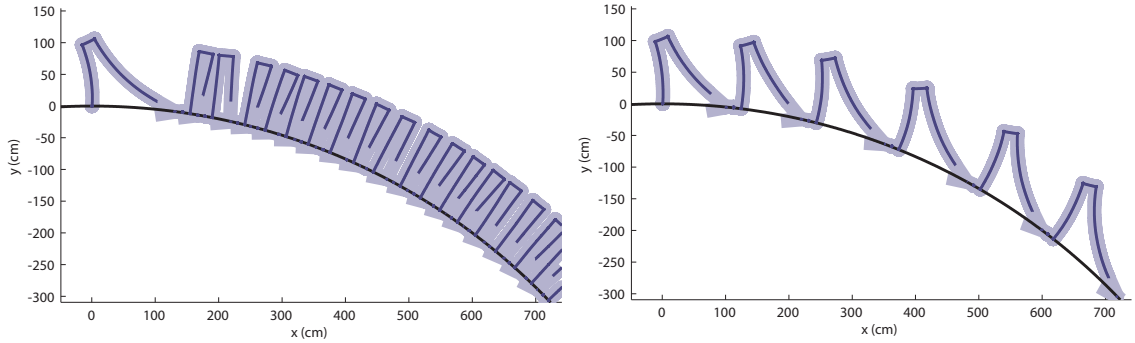
The Segbot is initially miscalibrated, simulating a step change in the effective parameters. We predict that the Segbot will initially perform poorly, but after several online calibration iterations will match the desired trajectory. Figure 8.12 shows that the robot is initially badly calibrated, which causes poor coverage. The parameters converge to the effective parameters due to successful calibration. The parameter convergence is shown in Figure 8.13.



(a) Concave boundary with $R = 1000$ cm



(b) Concave boundary with $R = 300$ cm



(c) Convex boundary with $R = 1000$ cm

Figure 8.9: Online calibration (left) finds an effective parameter set for constant curvature boundaries in the presence of process noise and provides better coverage than without calibration (right). This also demonstrates an approximation for arbitrary smooth boundaries.

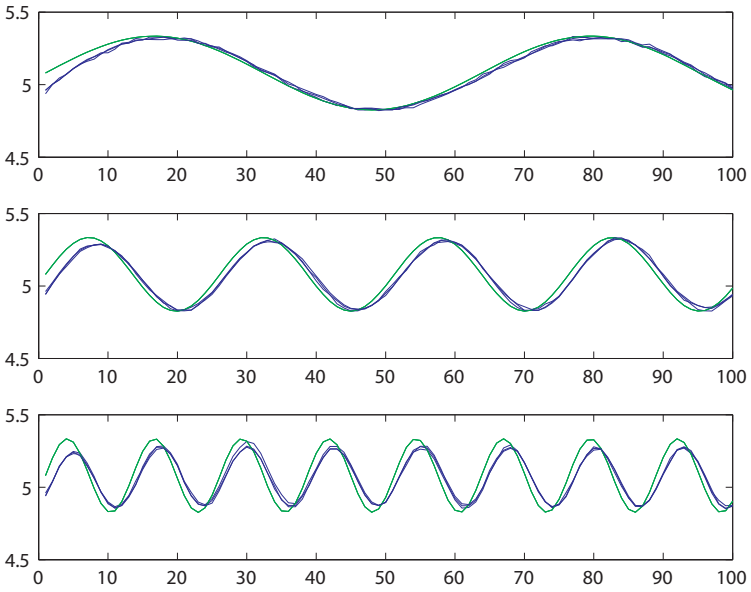


Figure 8.10: Tracking time-varying parameters. Actual parameters vary with an amplitude of 10% from the nominal values for three frequencies between 1 cycle per 15 iterations and 1 cycle per 60 iterations.

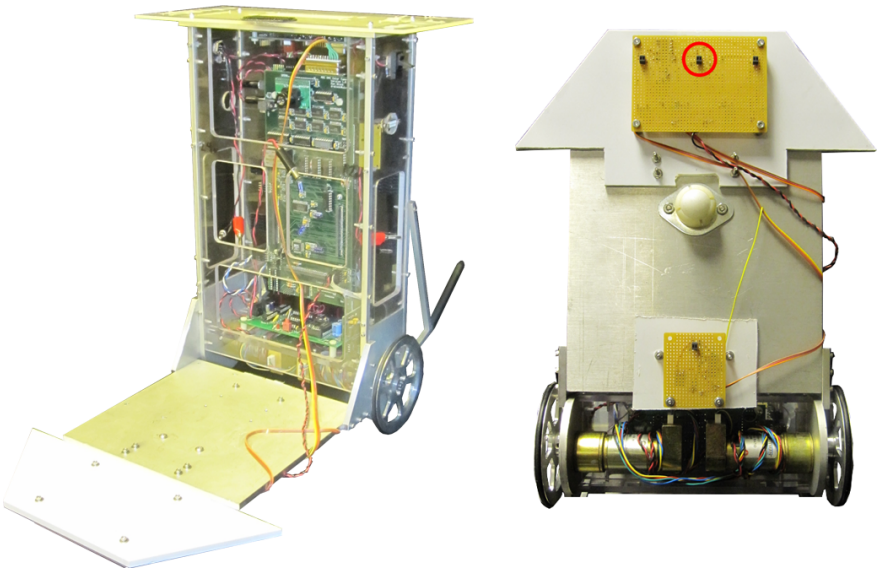


Figure 8.11: UIUC Segbot used for hardware validation. The reflective sensor used in experiments is on the underside of the robot, highlighted by a red circle.

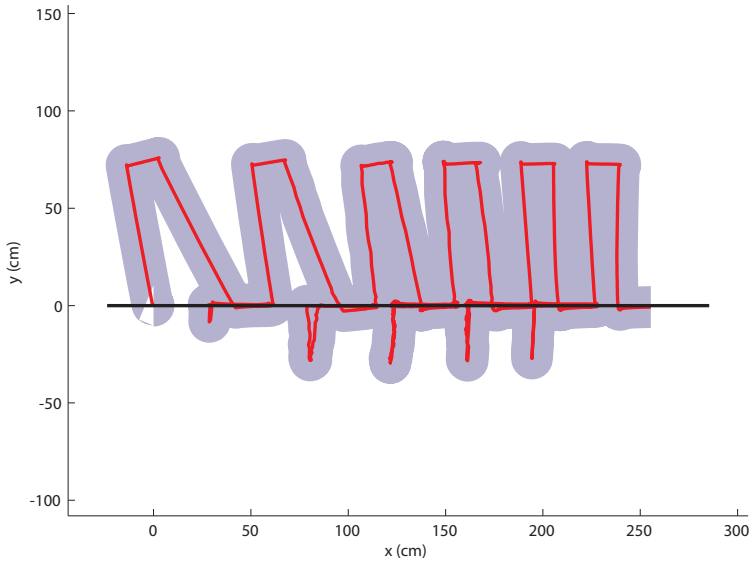


Figure 8.12: Experimental data from UIUC Segbot showing online calibration from initially miscalibrated parameters. The robot learns its effective parameters in six iterations. A robot with dynamic parameters shows improved coverage performance when well-calibrated.

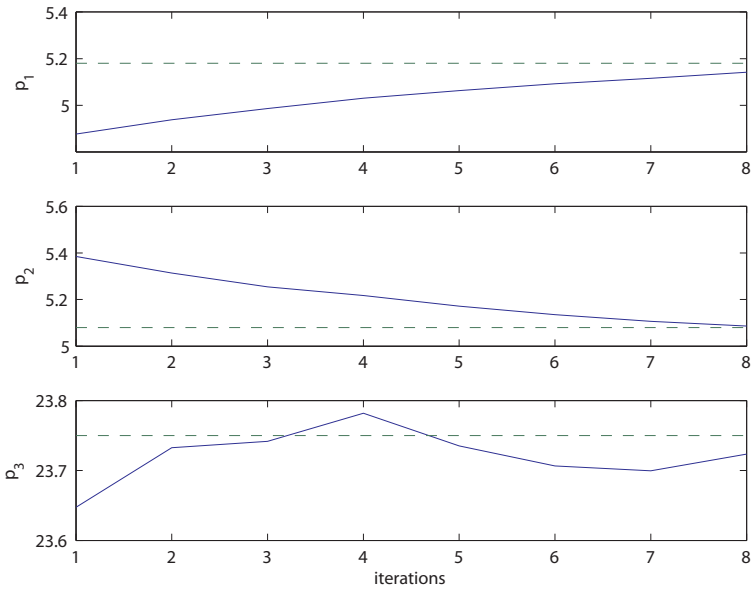


Figure 8.13: Three initially miscalibrated parameters converge to the effective values. This is validated by the coverage shown in Figure 8.12, which successfully implements a rectangular coverage path.

Chapter 9

Conclusion

The problem of robot coverage is important for a variety of military, industrial, and domestic applications. Provably complete methods exist under the assumption that the robot follows the path with negligible error. Under signficant uncertainty in sensing and actuation, it becomes unclear what problem we are trying to solve. We made the problem concrete by adopting a PAC measure of performance. We applied it to a simple system with uncertainty and showed that for a robot commanded to cover a rectangular region, following a square wave path with passes parallel to the short axis performs better than following a square wave path with passes parallel to the long axis.

This result inspired the design of a feedback policy, the virtual robot, which steered a square wave path throughout the free space. The virtual robot policy decoupled local and global coverage planning. We used a classical coverage planner, the boustrophedon decomposition, to plan a global path for the virtual robot—assumed to track the trajectory with negligible error. As the virtual robot tracked its trajectory, it incrementally generated a local path for the robot using a square wave motion primitive. We described in detail the models, calibration methods, and estimation and control processes for both the robot and the virtual robot. The virtual robot policy was implemented in simulation for a robot with uncertainty and yielded a higher fraction covered than a standard LQG strategy.

Finally, we built upon the virtual robot policy and considered operation specifically along the boundary. We introduced a method of simultaneous calibration and coverage for a robot with boundary sensors. In particular, the robot performed switchback trajectories in which it began at the boundary, commanded a set of inputs, and returned to the boundary. Through an analysis of observability, we chose a sequence of trajectories which yielded a robust estimation in parameters. Our algorithm was validated in simulation and hardware experiment.

Future work should express the PAC measure in functional form. The PAC measure should be used to optimize other policy parameters such as path spacing and overshoot and should adapt classical planners to systems with uncertainty by optimizing over parameters. The virtual robot should be rigorously tested against a random strategy and an LQG con-

troller under varying uncertainties and environments. A classical approach should be taken to develop the virtual robot state estimator instead of our heuristic approach. The PAC measure should also make clear the trade-off between calibration and coverage. By defining a cost function with a performance objective and the condition number, an optimal sequence of trajectories can be computed for the online calibration algorithm. Finally, future work should consider our use of the performance measure to develop local strategies and apply this to develop global strategies that are robust to uncertainty in sensing and actuation.

Appendix A

Derivations

A.1 Kinematic Model Derivation

The kinematic model is used in reconstructing the mobile robot configuration, $\boldsymbol{x} = [x, y, \theta]^T$ from the encoder measurements at the wheels. This process, known as *odometry* or *dead-reckoning*, starts from a known configuration and either uses direct measurements from the encoders or performs a discrete-time integration of wheel velocities to track angular displacements of each wheel.

A multitude of representations of such kinematics models appear in literature. Some are similar, but in different formats; others differ in algorithm. The main difference is typically the assumption that straight line motion and turning motion are independent. In [46], Wang geometrically derives the following model:

$$\begin{aligned} x_k &= x_{k-1} + \frac{\sin(\Delta\theta_k/2)}{\Delta\theta_k/2}\Delta d_k \cos(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ y_k &= y_{k-1} + \frac{\sin(\Delta\theta_k/2)}{\Delta\theta_k/2}\Delta d_k \sin(\theta_{k-1} + \frac{\Delta\theta_k}{2}) \\ \theta_k &= \theta_{k-1} + \Delta\theta_k. \end{aligned} \tag{A.1}$$

where Δd and $\Delta\theta$ represent incremental translational and rotational displacements. Here we adopt the following mathematical format:

$$\frac{\sin(\Delta\theta_k/2)}{\Delta\theta_k/2} = \mathrm{sinc}(\Delta\theta_k/2)$$

In most sources, this *sinc* term is rounded to unity [54], [59], and [60]. As shown in [46], this is still a good approximation because the encoder sampling rate is so high comapred to the wheel velocities along with the following property of *sinc* function: $(\mathrm{sinc}(x) \to 1 \mid x \to 0)$. However, presented here is a different method of arriving at the same kinematic model to

provide more clarity. We begin with the continuous-time differential equations:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{A.2}$$

The goal is to create a mapping from incremental change in heading $\Delta\theta$ and translation Δd to change in configuration $[\Delta x, \Delta y, \Delta\theta]$. The $y-$term will now be put into a more useful format and integrated for one time step: $0 \to \Delta t$.

$$\begin{aligned} \dot{y} &= v\sin(\omega\Delta t + \theta_0) \\ y - y_0 &= \int_0^t v\sin(\omega\Delta t + \theta_0)dt \\ &= \frac{v}{\omega}\left[-\cos(\omega\Delta t + \theta_0)\right]_0^t \\ &= \frac{v}{\omega}\left[-\cos(\omega\Delta t + \theta_0) + cos(\theta_0)\right] \end{aligned} \tag{A.3}$$

but it is known that for a single time increment,

$$\omega\Delta t = \Delta\theta$$

And by splitting up this term, (A.3) can be rewritten

$$= \frac{v}{\omega}\left[-\cos\left(\underbrace{\left(\theta_0 + \frac{\Delta\theta}{2}\right) + \frac{\Delta\theta}{2}}_{\theta_0+\Delta\theta}\right) + \cos\left(\underbrace{\left(\theta_0 + \frac{\Delta\theta}{2}\right) - \frac{\Delta\theta}{2}}_{\theta_0}\right)\right]$$

And expanding this result using the cosine and sine angle addition rules yields

$$\begin{aligned} = \frac{v}{\omega}\Bigg\{ &- \cancel{\left[\cos\left(\theta_0 + \frac{\Delta\theta}{2}\right)\cos\left(\frac{\Delta\theta}{2}\right)\right]} - \sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)\sin\left(\frac{\Delta\theta}{2}\right)\Bigg] \\ &+ \cancel{\left[\cos\left(\theta_0 + \frac{\Delta\theta}{2}\right)\cos\left(\frac{\Delta\theta}{2}\right)\right]} + \sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)\sin\left(\frac{\Delta\theta}{2}\right)\Bigg]\Bigg\} \end{aligned}$$

which simplifies to

$$= \frac{v}{\omega}\left[2\sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)\sin\left(\frac{\Delta\theta}{2}\right)\right]$$

where this can be multiplied by $\frac{\Delta\theta}{\Delta\theta}$ and where $2 = \frac{1}{1/2}$. So,

$$= \frac{v}{\omega}\,\frac{\Delta\theta}{\Delta\theta}\,\frac{1}{1/2}\sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)\sin\left(\frac{\Delta\theta}{2}\right)$$

but $\omega\Delta t = \Delta\theta$ or $\frac{\Delta\theta}{\omega} = \Delta t$. So,

$$= v\Delta t\frac{sin\,(\Delta\theta/2)}{\Delta\theta/2}\sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)$$

$$y - y_0 = v\Delta t\,\mathrm{sinc}\,(\Delta\theta/2)\sin\left(\theta_0 + \frac{\Delta\theta}{2}\right) \tag{A.4}$$

And finally with $\Delta t \to \mathrm{dt}\,,$

$$y = y_0 + v\,\mathrm{dt}\,\mathrm{sinc}\,(\Delta\theta/2)\sin\left(\theta_0 + \frac{\Delta\theta}{2}\right)$$

Similarly for the $x-$term,

$$x = x_0 + v\,\mathrm{dt}\,\mathrm{sinc}\,(\Delta\theta/2)\cos\left(\theta_0 + \frac{\Delta\theta}{2}\right)$$

The heading equation remains unchanged and vt is equivalent to Δd_i. Thus, the same result as [46] is produced, now analytically. Note, that if using these equations offline, that some software with the built in *sinc* function multiply its argument by π and define it $\mathrm{sinc}\,(x) = \sin(\pi x)/(\pi x)$.

A.2 Derivation: Propogating Error Model

The "propogating modeling uncertainty" covariance matrix assumes that the odometric parameters are known with some uncertainty; see the Equations in (5.32). The resulting

filter input $u = [\Delta d^* \ \Delta\theta^*]^T$ is then

$$\begin{aligned}
\Delta d^* &= \frac{(r_R + \tilde{\delta} r_R)\alpha_R + (r_L + \tilde{\delta} r_L)\alpha_L}{2} \\
&= \Delta d + \frac{\tilde{\delta} r_R \alpha_R + \tilde{\delta} r_L \alpha_L}{2} \\
\Delta\theta^* &= \frac{(r_R + \tilde{\delta} r_R)\alpha_R - (r_L + \tilde{\delta} r_L)\alpha_L}{b + \tilde{\delta} b} \\
&= \frac{b}{b + \tilde{\delta} b}\left(\Delta\theta + \frac{\tilde{\delta} r_R \alpha_R + \tilde{\delta} r_L \alpha_L}{b}\right)
\end{aligned}$$

From these, the worst case values become

$$\begin{aligned}
\Delta d^*_{max} &= \Delta d + \frac{|\tilde{\delta} r_R \alpha_R| + |\tilde{\Delta} r_L \alpha_L|}{2} \\
\Delta d^*_{min} &= \Delta d - \frac{|\tilde{\delta} r_R \alpha_R| + |\tilde{\delta} r_L \alpha_L|}{2} \\
\Delta\theta^*_{max} &= \frac{b}{b - \text{sign}(\Delta\theta)|\tilde{\delta} b|}\left(\Delta\theta + \frac{|\tilde{\delta} r_R \alpha_R| + |\tilde{\delta} r_L \alpha_L|}{b}\right) \\
\Delta\theta^*_{min} &= \frac{b}{b - \text{sign}(\Delta\theta)|\tilde{\delta} b|}\left(\Delta\theta - \frac{|\tilde{\delta} r_R \alpha_R| + |\tilde{\delta} r_L \alpha_L|}{b}\right)
\end{aligned}$$

Thus, the input vector uncertainties are

$$\delta\Delta d = \frac{\Delta d^*_{max} - \Delta d^*_{min}}{2}$$

$$\delta\Delta d = |\tilde{\delta} r_R \alpha_R| + |\tilde{\delta} r_L \alpha_L| \tag{A.5}$$

and

$$\delta\Delta\theta = \frac{\Delta\theta^*_{max} - \Delta\theta^*_{min}}{2}$$

$$\delta\Delta\theta = \frac{2b}{b^2 - \Delta b^2}\left(|\tilde{\delta} b \Delta\theta| + |\tilde{\delta} r_R \alpha_R| + |\tilde{\delta} r_L \alpha_L|\right) \tag{A.6}$$

The matrix describing squared uncertainties of Δd^* and $\Delta\theta^*$ for substitution into (5.31) is shown in Equation (5.35).

A.3 Odometry Covariance Matrix: an alternate computation

The odometry error covariance matrix is a (3×3) matrix that is typically based off of the motion model, i.e., the process noise model. However, it may also be determined by

analyzing the final configurations of *many* straight line trajectories. The differences between initial and final configurations, denoted with a d prefix, are found as computed by odometry, denoted CAN, and as measured by an external sensor, denoted ES. The odometry error is then computed by subtracting ES in the numerator of (A.7). Dividing by the path length, approximated by the summation of the incremental Euclidean distances, makes the errors per-unit-length, which is applicable to computations for incremental movements of varying size. Assembling this data for n trajectories produces an $(n \times 3)$ matrix D_{CAN-ES}.

$$\frac{\begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix}_{CAN} - \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix}_{ES}}{\sum \sqrt{dx_{CAN,i}^2 + dy_{CAN,i}^2}} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix}_{CAN-ES} \tag{A.7}$$

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta \theta_1 \\ & \vdots & \\ \Delta x_n & \Delta y_n & \Delta \theta_n \end{bmatrix} = D_{CAN-ES} \tag{A.8}$$

The odometry error covariance matrix is the covariance of D_{CAN-ES} which may be manually computed using (A.9).

$$\Sigma_{ij} = \operatorname{cov}(X_i, X_j) = E\left[(X_i - \mu_i)(X_j - \mu_j)\right] \tag{A.9}$$

where

$$\mu_i = E(X_i)$$

Appendix B

Heuristic Controllers

We present three heuristic controllers for tracking different representations of trajectories. We begin with a proportional controller for movement along straight lines and zero-point turns. This is useful in generating a reference trajectory. We then present a path following algorithm if a robot is given a reference path. Finally, we present a point-to-point controller for a robot to visit a set of waypoints.

B.1 Proportional Path Control

Many high level controllers assume that the robot applies locally referenced inputs to move it along trajectories or motion primatives. This is primarily useful in test scenarios where global information is not needed. In this case, it is simple to provide a list of desired changes in state. Here, we present a simple controller to execute a list of the following pairs.

1. $(\Delta\theta, \omega_{max})$. $\Delta\theta$ is the desired change in heading relative to the current heading and ω_{max} is the maximum angular velocity.

2. $(\Delta d, v_{max})$. Δd is the desired forward distance to travel and v_{max} is the maximum magnitude of path velocity.

To avoid slippage, proportional controllers command forward velocity and turn rate for each maneuver, i.e., straight line or ZPT. Figure 4 states the robot trajectory control algorithm.

We use the following notation: d^* is the total target distance to travel along the maneuver, d is the distance traveled along the maneuver, $\Delta\theta^*$ is the desired change in heading, θ_i is the current heading estimate, θ_0 is the heading estimate at the start of the maneuver. For straight line maneuvers, $\omega = 0$ and for ZPTs, $v = 0$. The *atan2* function is used to compute is used to find θ_{error} the shortest angle between the current heading and the desired heading. The robot commands are bounded by $v_{\max}$ and $\omega_{\max}$ in the function Check_Input_Saturation. Values used for proportional control are in Table B.1. The minimum errors and end-condition for each maneuver are θ_{thresh} and d_{thresh}.

Algorithm 4 $[v, \omega, \text{NextManeuver}] = \text{PROPORTIONAL\_SL\_OR\_ZPT}$ $(\text{TurnManeuver}, \{(\Delta\theta^*, \theta_0, \theta_i), (d^*, d)\})$

NextManeuver = 0
if TurnManeuver **then**
  $\theta_{error} = \text{atan2}(\sin(\Delta\theta^* - \theta_i + \theta_0), \cos(\Delta\theta^* - \theta_i + \theta_0))$
  $\omega = K_{p,\omega}\,\theta_{error}$
  if $|\theta_{error} < \theta_{thresh}|$ **then**
    NextManeuver = 1
  end if
else
  $d_{error} = d^* - d$
  $v = K_{p,v} d_{err}$
  if $|d_{err}| < d_{thresh}$ **then**
    NextManeuver = 1
  end if
end if
$(v, \omega) = \text{Check\_Input\_Saturation}(v_{max}, \omega_{max}, v, \omega)$
Coupled_PI_Control(v, ω) // Low level wheel angular velocity control

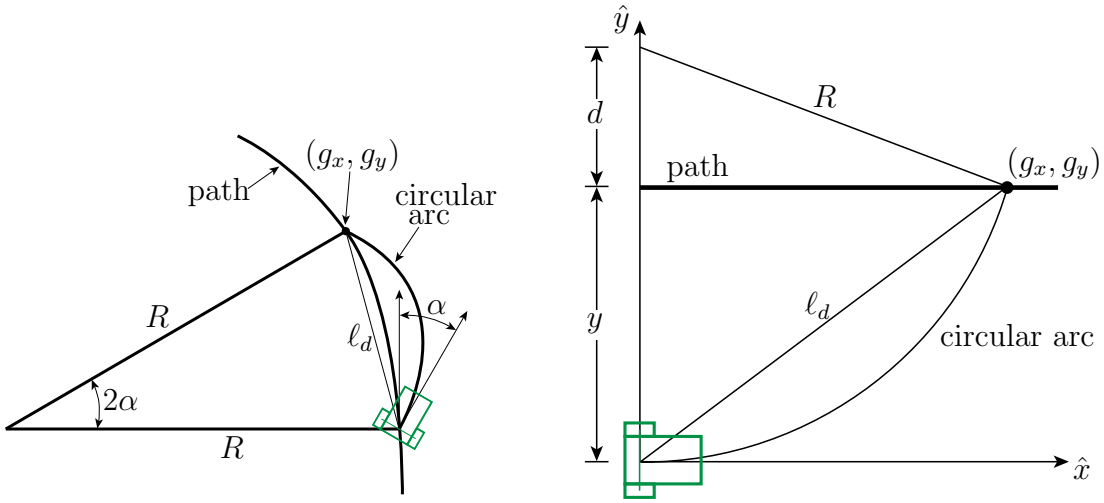
The velocity and turn commands are then sent to a lower level coupled PI controller which commands wheel angular velocities as discussed in Section 6.2.1.

B.2 Path Following

One method of path following is pure pursuit which involves geometrically calculating the curvature of a circular arc that connects the current vehicle position to a goal point ahead of the vehicle on the desired path. The goal point (g_x, g_y) is determined by a look-ahead

Table B.1: Values for Proportional Path Control

Description	Symbol	Value
max. path velocity	$v_{\max}$	45 cm/s
max. turn rate	$\omega_{\max}$	$\frac{\pi}{3}$ rad/s
proportional velocity constant	$K_{p,v}$	1.0
proportional turn constant	$K_{p,\omega}$	1.0
min. distance error	d_{thresh}	0.001
min. heading error	θ_{thresh}	0.001



(a) heading error (b) cross-track error

Figure B.1: Path following geometry.

distance ℓ_d. The curvature can relate to the vehicle's heading error, α, the difference in angle between the vehicle vector and the tangential path vector at the closest path point [61]. The curvature can also relate to the vehicle's cross-track error, e_c, the distance between the current vehicle position and the closest path point [62].

The method here combines both approaches mentioned above. Curvature, $\gamma = 1/R$, is related to both:

1. heading error, α
2. cross-track error, e_c

Deriving the former, the law of sines can be applied to Figure B.1(a).

$$\begin{aligned}
\frac{\ell_d}{\sin(2\alpha)} &= \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)} \\
\frac{\ell_d}{2\sin(\alpha)\cos(\alpha)} &= \frac{R}{\cos(\alpha)} \\
\frac{\ell_d}{\sin(\alpha)} &= 2R \\
\gamma_h &= \frac{2\sin(\alpha)}{\ell_d}
\end{aligned}$$

Deriving the latter, from Figure B.1(b),

$$g_x^2 + g_y^2 = \ell_d^2 \tag{B.1}$$

$$y + d = r \tag{B.2}$$

The origin is located at the vehicle wheel axel implying that vehicle coordinates are used here. Thus, the goal point now has the coordinates (x, y). Equation (B.1) states that the locus of possible goal points for the vehicle lie on a circle of radius ℓ_d and Equation (B.2) states that the radius of curvature intersects the y-axis with offset d. It is also clear then that the goal point is simply related to the radius of curvature:

$$x^2 + y^2 = r^2$$

This leads to the following series of equations.

$$\begin{aligned} d &= r - y \\ x^2 + (r - y)^2 &= r^2 \\ x^2 r^2 - 2ry + y^2 &= r^2 \\ 2ry &= \ell_d^2 \\ r &= \frac{\ell_d^2}{2y} \\ \gamma_{ct} &= \frac{2y}{\ell_d^2} \end{aligned}$$

Because it is known that curvature is inversely proportional to the radius of curvature, then obviously the instantaneous turn command is

$$w = v\gamma$$

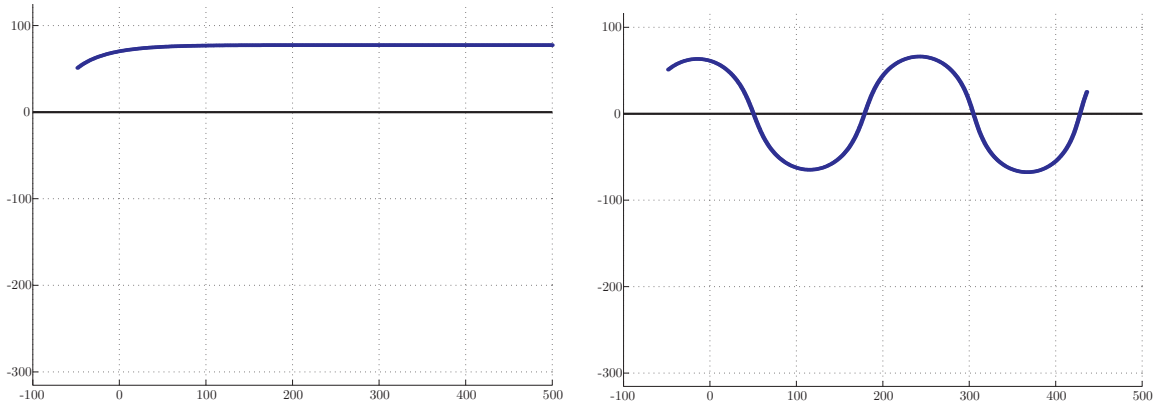
With the expressions for curvature and the relation to the turn command, the algorithm for pure pursuit is outlined in Figure B.2.

When applying each of these control policies, neither properly followed the desired path. Using the same initial conditions, the results for the heading error pursuit are shown in Figure B.3(a) and the results for the cross-track error pursuit are shown in Figure B.3(b). The algorithm that corrects for heading error clearly does so, but leaves some steady-state cross-track error. The overshoot that occurs in the cross-track-dependent policy is propagates. This is expected beacuse as the vehicle approaches the desired path, the turn command

1. determine state, $\boldsymbol{x}$
2. find goal point, (g_x, g_y)
3. calculate curvature, γ
4. send respective turn command, ω
5. update state

Figure B.2: Path following (pure pursuit) algorithm

diminishes, allowing the vehicle to move past it.



(a) heading error (b) cross-track error

Figure B.3: Pure pursuit with different curvature dependence.

Combining the two approaches results in a clean path following. See Figure B.4. This is intuitive to have the turn command correct for both cross-track and heading error. The curvature used in this approach is as follows:

$$\gamma = \gamma_h + \gamma_{ct}$$
$$\gamma = \frac{2\sin(\alpha)}{\ell_d} + \frac{2y}{\ell_d^2}$$

B.3 Point-to-Point

Another scheme of trajectory controllers is point-to-point - where the robot is given a set of desired waypoints to visit. We give a heuristic implementation here. Based on the diagrams

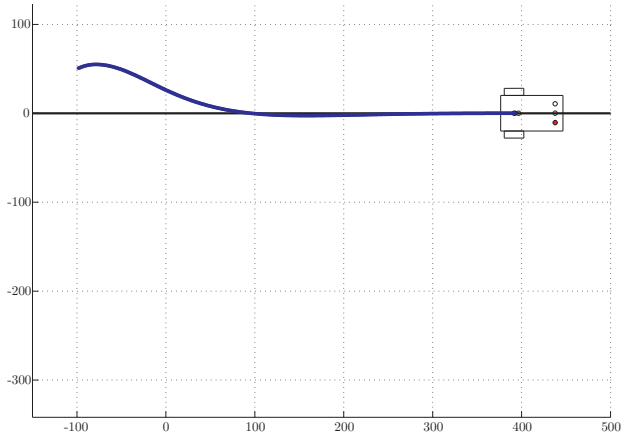


Figure B.4: Path following (pure pursuit) by using both heading error and cross-track error. The agent coverges to the target path.

in Figure B.5, the idea is to have a robot travel from one point to another (in the order in which they are given) to within some distance r of each waypoint. For tighter error bounds, this number should be smaller; we choose $r = 0.001$. In order to move from point to point, the robot executes the algorithm in Figure 6.

We implement this as a function with inputs (x_1, y_1, x_2, y_2) and outputs $(v, \omega, \text{TargetReached})$, where (x_1, y_1) is the robot's current position and (x_2, y_2) is the next waypoint, i.e., the robot's desired position, v is the path velocity, ω is the turn rate, and ReachedTarget is a Boolean that is TRUE when the robot is within r of the desired position and false otherwise. The heading θ is bounded to the interval $[-\pi, \pi]$ and the trajectory angle $\alpha \in [-\pi, \pi]$ is the absolute angle to the desired position. The heading error $\theta_{err} \in [-\pi, \pi]$ is then the difference between α and θ. The output velocity and turn rate are proportional to the distance to travel d and the heading error.

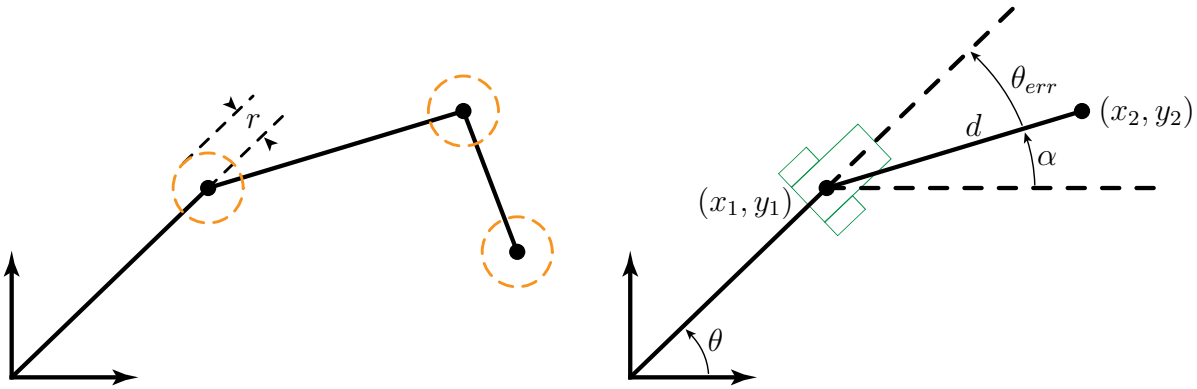


Figure B.5: Point-to-point controller geometry.

Algorithm 6 Point-to-point algorithm

if $\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} < r$ **then**
  $\theta \in [-\pi, \pi]$ // current heading
  $d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$ // distance to travel
  $\alpha = \text{atan2}\,(y_1 - y_0, x_1 - x_0)$ // trajectory angle
  $\theta_{err} = \alpha - \theta$ // heading error
  $\theta_{err} \in [-\pi, \pi]$ // find shortest angle
  $v = K_{p,v}\, d$ // proportional velocity control
  $\omega = K_{p,\omega}\, \theta_{err}$ // proportional heading control
  ReachedTarget = **false**
else
  ReachedTarget = **true**
end if

Appendix C

Supplemental Files

Two video files are included with this thesis:

1. `robot_coverage_with_uncertainty.mp4`
2. `online_calibration.m4v`

The first video summarizes the work from Section 2.2. It formulates the problem of coverage under uncertainty by introducing our PAC measure of performance and shows an application of this measure. The second video shows the hardware implementation of the online calibration algorithm from Section 8.

References

[1] E. M. Arkin and R. Hassin, "Approximation algorithms for the geometric covering salesman problem," *Discrete Applied Mathematics*, vol. 55, no. 3, pp. 197–218, 12 1994. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYW-45DHW0G-1/2/b11923f171c916c04d9f3b443c199675

[2] E. M. Arkin, S. P. Fekete, and J. S. B. Mitchell, "Approximation algorithms for lawn mowing and milling," *Computational Geometry*, vol. 17, no. 1-2, pp. 25–50, 1997.

[3] A. Zelinsky, R. Jarvis, J. C. Byrne, and S. Yuta, "Planning paths of complete coverage of an unstructured environment by a mobile robot," in *In Proceedings of International Conference on Advanced Robotics*, 1993, pp. 533–538.

[4] H. Choset, "Coverage of known spaces: The boustrophedon cellular decomposition," *Autonomous Robots*, vol. 9, pp. 247–253, 2000.

[5] ——, "Coverage for robotics – a survey of recent results," *Annals of Mathematics and Artificial Intelligence*, vol. 31, no. 1-4, pp. 113–126, 2001.

[6] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull, "Morse decompositions for coverage tasks," *I. J. Robotic Res.*, vol. 21, no. 4, pp. 331–344, 2002.

[7] E. U. Acar and H. Choset, "Robust sensor-based coverage of unstructured environments," in *Proceedings of IROS 2001*, Oct. 2001, pp. 61– 68.

[8] E. U. Acar, H. Choset, and P. N. Atkar, "Complete sensor-based coverage with extended-range detectors: a hierarchical decomposition in terms of critical points and Voronoi diagrams," in *Proceedings of IROS 2001*, 2001, pp. 1305–1311.

[9] E. U. Acar, H. Choset, Y. Zhang, and M. J. Schervish, "Path planning for robotic demining: Robust sensor-based coverage of unstructured environments and probabilistic methods," *I. J. Robotic Res.*, vol. 22, no. 7-8, pp. 441–466, 2003.

[10] E. Acar, H. Choset, and J. Y. Lee, "Sensor-based coverage with extended range detectors," *Robotics, IEEE Transactions on*, vol. 22, no. 1, pp. 189 – 198, feb. 2006.

[11] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementation.* The MIT Press, 2005.

[12] J. Pearce, B. Powers, C. Hess, P. Rybski, S. Stoeter, and N. Papanikolopoulos, "Using virtual pheromones and cameras for dispersing a team of multiple miniature robots," *Journal of Intelligent & Robotic Systems*, vol. 45, pp. 307–321, 2006.

[13] J. Svennebring and S. Koenig, "Building terrain-covering ant robots: A feasibility study," *Autonomous Robots*, vol. 16, no. 3, pp. 313–332, 2004.

[14] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Autom.*, vol. 2, no. 1, pp. 14–23, 1986.

[15] Z. L. Cao, Y. Huang, and E. L. Hall, "Region filling operations with random obstacle avoidance for mobile robots," *J. Robotic Systems*, vol. 5, pp. 87–102, 1988.

[16] R. De Carvalho, H. Vidal, P. Vieira, and M. Ribeiro, "Complete coverage path planning and guidance for cleaning robots," in *Industrial Electronics, 1997. ISIE '97., Proceedings of the IEEE International Symposium on*, vol. 2, Jul. 1997, pp. 677–682.

[17] D. W. Gage, "Many-robot MCM search systems," in *Proceedings of Autonomous Vehicles in Mine Countermeasures Symposium*, 1995, pp. 9–55.

[18] S. S. Ge and C. Fua, "Complete multi-robot coverage of unknown environments with minimum repeated coverage," in *Proceedings of ICRA 2005*, Apr. 2005, pp. 715–720.

[19] E. Gelenbe and Y. Cao, "Autonomous search for mines," *European Journal of Operational Research*, vol. 108, no. 2, pp. 319–333, 1998.

[20] Y. Guo and M. Balakrishnan, "Complete coverage control for nonholonomic mobile robots in dynamic environments," in *Proceedings of ICRA 2006*, May 2006, pp. 1704–1709.

[21] S. Hert, S. Tiwari, and V. Lumelsky, "A terrain-covering algorithm for an auv," *Autonomous Robots*, vol. 3, pp. 91–119, 1996.

[22] D. Kurabayashi, J. Ota, T. Arai, S. Ichikawa, S. Koga, H. Asama, and I. Endo, "Cooperative sweeping by multiple mobile robots with relocating portable obstacles," in *Proceedings of IROS 1996*, vol. 3, Nov. 1996, pp. 1472–1477.

[23] I. Latimer, D., S. Srinivasa, V. Lee-Shue, S. Sonne, H. Choset, and A. Hurst, "Towards sensor based coverage with robot teams," in *Proceedings of ICRA 2002*, vol. 1, 2002, pp. 961–967.

[24] C. Luo and S. Yang, "A real-time cooperative sweeping strategy for multiple cleaning robots," in *Proceedings of the 2002 IEEE International Symposium on Intelligent Control*, 2002, pp. 660–665.

[25] T. W. Min and H. K. Yin, "A decentralized approach for cooperative sweeping by multiple mobile robots," in *Proceedings of IROS 1998*, vol. 1, Oct. 1998, pp. 380–385.

[26] I. Rekleitis, A. New, and H. Choset, "Distributed coverage of unknown/unstructured environments by mobile sensor networks," in *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, L. Parker, F. Schneider, and A. Schultz, Eds. Springer Netherlands, 2005, pp. 145–155.

[27] J. Rogge and D. Aeyels, "Sensor coverage with a multi-robot system," in *Intelligent Control, 2007. ISIC 2007. IEEE 22nd International Symposium on*, Oct. 2007, pp. 71–76.

[28] P. Hertling, L. Hog, R. Larsen, J. Perram, and H. Petersen, "Task curve planning for painting robots. i. process modeling and calibration," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 2, pp. 324 –330, apr. 1996.

[29] J. S. Oh, Y. H. Choi, J. B. Park, and Y. Zheng, "Complete coverage navigation of cleaning robots using triangular-cell-based map," *Industrial Electronics, IEEE Transactions on*, vol. 51, no. 3, pp. 718 – 726, jun. 2004.

[30] S. Yang and C. Luo, "A neural network approach to complete coverage path planning," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, no. 1, pp. 718 – 724, feb. 2004.

[31] D. Conner, A. Greenfield, P. Atkar, A. Rizzi, and H. Choset, "Paint deposition modeling for trajectory planning on automotive surfaces," *Automation Science and Engineering, IEEE Transactions on*, vol. 2, no. 4, pp. 381 – 392, oct. 2005.

[32] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[33] T. Palleja, M. Tresanchez, M. Teixido, and J. Palacin, "Modeling floor-cleaning coverage performances of some domestic mobile robots in a reduced scenario," *Robot. Auton. Syst.*, vol. 58, no. 1, pp. 37–45, 2010.

[34] P. G. Hall, *Introduction to the theory of Coverage Processes*. New York, New York: Wiley Series in Probability and Mathematical Statistics, 1988.

[35] W. Huang, "Optimal line-sweep-based decompositions for coverage algorithms," in *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, 2001.

[36] J. Borenstein and L. Feng, "Umbmark-a method for measuring, comparing and correcting dead-reckoning errors in mobile robots," University of Michigan, Tech. Rep., 1994.

[37] M. Erdmann, "Using backprojections for fine motion planning with uncertainty," *I. J. Robotic Res.*, vol. 5, no. 1, pp. 19–45, 1986.

[38] T. Lozano-pérez, M. T. Mason, and R. H. Taylor, "Automatic synthesis of fine-motion strategies for robots," *I. J. Robotic Res.*, pp. 60–99, 1984.

[39] L. D. Stone, "Search theory: A mathematical theory for finding lost objects," *Mathematics Magazine*, vol. 50, no. 5, pp. 248–256, 1977.

[40] H. Solomon, *Geometric probability*. Philadelphia : Society for Industrial and Applied Mathematics, 1978.

[41] A. M. Kellerer, "On the number of clumps resulting from the overlap of randomly placed figures in a plane," *J. Applied Probability*, vol. 20, no. 1, pp. 126–135, Mar. 1983.

[42] I. C. on Mine Clearance Technology, "Report of the international conference on mine clearance technology," in *International Conference on Mine Clearance Technology*, Elsinore, Denmark, Jul. 1996.

[43] L. G. Valiant, "A theory of the learnable," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 436–445.

[44] D. Haussler, "Probably approximately correct learning," in *Proceedings of the Eighth National Conference on Artificial Intelligence*. AAAI Press, 1990, pp. 1101–1108.

[45] J. L. Jones and P. R. Mass, "Method and system for multi-mode coverage for an autonomous robot," Sep. 2007.

[46] C. M. Wang, "Location estimation and uncertainty analysis for mobile robots," *IEEE*, pp. 1230–1235, 1988.

[47] L. Kleeman, "Odometry error covariance estimation for two wheel robot vehicles," Intelligent Robotics Research Centre, Department of Electrical and Computer Systems Engineering, Monash University, Tech. Rep. MECSE-95-1, 1995.

[48] A. Martinelli, N. Tomatis, and R. Siegwart, "Simultaneous localization and odometry self calibration for mobile robot," *Auton Robot*, vol. 22, pp. 75–85, 2007.

[49] K. S. Chong and L. Kleeman, "Accurate odometry and error modelling for a mobile robot," *MECSE*, April 1996.

[50] ——, "Accurate odometry and error modelling for a mobile robot," *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, vol. 4, pp. 2783–2788, April 1997.

[51] A. Kelly, "General solution for linearized stochastic error propagation in vehicle odometry," *Preprints 15th IFAC World Congress*, July 2002.

[52] G. Antonelli and S. Chiaverini, "Linear estimation of the physical odometric parameters for differential-drive mobile robots," *Auton Robot*, vol. 23, pp. 59–68, 2007.

[53] A. Martinelli and R. Siegwart, "Estimating the odometry error of a mobile robot during navigation," *European Conference on Mobile Robotics (ECMR 2003)*, 2003.

[54] T. D. Larsen, M. Bak, N. A. Andersen, and O. Ravn, "Location estimation for an autonomously guided vehicle using an augmented kalman filter to autocalibrate the odometry," *FUSION98 SPIE Conference*, July 1998.

[55] T. D. Larsen, "Optimal fusion of sensors," Ph.D. dissertation, Department of Automation, Technical University of Denmark, September 1998.

[56] D. F. Sebastian Thrun, Wolfram Burgard, *Probabilistic Robotics*. The MIT Press, 2006.

[57] D. Johnson, "Mechanical design, construction, and control of an autonomous segway operator," Master's thesis, University of Illinois at Urbana-Champaign, 2005.

[58] R. Hermann and A. J. Krener, "Nonlinear controllability and observability," *IEEE Transactions on Automatic Control*, vol. AC-22, no. 5, pp. 728–740, 1977.

[59] A. I. Eliazar and R. Parr, "Learning probabilistic motion models for mobile robots," *Proc. 21st ICML'04*, pp. 32–39, July 2004.

[60] J. Teddy N. Yap and C. R. Shelton, "Simultaneous learning of motion and sensor model parameters for mobile robots," *2008 IEEE International Conference on Robotics and Automation*, pp. 2091–2097, May 2008.

[61] J. M. Snider, "Automatic steering methods for autonomous automobile path tracking," Robotics Institute, Carnegie Mellon University, technical CMU-RI-TR-09-08, 2009.

[62] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," The Robotics Institute, Carnegie Mellon University, technical CMU-RI-TR-92-01, 1992.